Estructura Genérica para Algoritmos de Ruteo de Vehículos

Informe Final

Luis Ignacio Alonzo
María Fernanda Burgueño

Resumen

En este documento presentamos la documentación de un proyecto de Taller V. En el mismo, introducimos al lector al Problema de Ruteo de Vehículos (VRP), un área en creciente desarrollo tanto por su interés económico (para las empresas que se dedican a la recolección/distribución de bienes y servicios), como académico (dada la dificultad que presenta el problema); y a algunas posibles soluciones. La propuesta, realizada por el Área de Optimización de la Empresa Ingenieros Consultores Asociados (ICA) surge de la necesidad de contar con una estructura genérica que permita simplificar la implementación de algoritmos que resuelven el VRP, necesidad que se confirma a partir de la investigación bibliográfica realizada para este proyecto. Se presenta además el análisis y diseño de una estructura, así como también un plan de testeo para la misma y los resultados del testeo de la estructura diseñada e implementada. Por último se presentan las conclusiones del trabajo (tanto a nivel de diseño, como a nivel de implementación) y posibles caminos por los que puede continuarse el trabajo realizado.

Índice

Introducción

PROBLEMA

Introducción

Contexto

Este trabajo es realizado por un grupo de Taller V a solicitud de la Empresa Ingenieros Consultores Asociados (ICA), cediendo a la misma todos los derechos sobre el producto resultante.

Responsables

ICA: Ing. José Comas, Area Optimización
InCo: Ing. Omar Viera, Responsable Académico

Objetivos del Proyecto

El proyecto consiste en el desarrollo de una Estructura de Datos que permita utilizarla como una biblioteca para simplificar la tarea del programador de algoritmos destinados a resolver el Problema de Ruteo de Vehículos (VRP por sus siglas en inglés) y sus extensiones. Dicha Estructura deberá ser genérica de modo que permita solucionar cualquier VRP, independientemente del problema en particular y del algoritmo elegido para su solución.

VRP

Este problema consiste en "determinar un conjunto de rutas de costo mínimo para una flota homogénea de vehículos que sirve a un conjunto de clientes dispersos geográficamente".
Es un problema de interés tanto académico como privado. En lo académico por tratarse de un problema NP-Duro[1], en lo privado por los enormes costos que tiene para las Empresas dedicadas a la Distribución/Recolección por concepto de fletes.

Estructura

Si bien durante las dos ultimas décadas se han realizado grandes avances en el desarrollo de métodos de solución del VRP tanto a nivel de los llamados Métodos Exactos (que obtienen una solución optima pero en tiempo de calculo inaceptable) como a nivel de Métodos Heurísticos (que obtienen una solución "aceptable" rápidamente), no ha habido un desarrollo equivalente de Estructuras de Datos genéricas que soporten algoritmos para la solución de cualquier VRP.

[1] Ver Capítulo Introducción al Problema de Ruteo de Vehículos

A partir de lo investigado para llevar a cabo este proyecto, tanto a nivel bibliográfico como en Internet, el único ejemplo de esfuerzos en este sentido es el producto Netengine de ESRI.

Desarrollo de los objetivos

- Desarrollar la citada estructura (Análisis, Diseño, Implementación y Testeo)
- Realizar la implementación en un lenguaje NO orientado a Información Geográfica, de modo que la Empresa pueda, a posteriori comparar estos resultados con los obtenidos por otro grupo, que realizará la implementación en un lenguaje orientado a Información Geográfica. Se elige C++ como lenguaje de implementación
- Se desea poder ingresar la información por capas temáticas (por ejemplo calles, avenidas, puentes, etc.), de modo de poder luego tomar en cuenta todas o solo algunas de dichas capas.
- Como producto intermedio, la Empresa solicita la implementación de una interface entre la salida de ARCVIEW (Sistema de Información Geográfica) y un archivo de texto con un formato dado, de modo de que la Estructura de Datos a implementar sea independiente del Sistema de Información Geográfica. Este producto intermedio puede considerarse como un producto en si mismo, dado que soluciona el problema de interacción con ARCVIEW, permitiendo levantar un mapa, originalmente digitalizado en ARCVIEW desde otros lenguajes de programación o Sistemas de Información Geográfica.

Desarrollo de la Estructura

Luego de una búsqueda bibliográfica y de varias reuniones con el Representante de la Empresa , se decide representar la estructura como un Grafo dirigido con Información y una lista de costos en los nodos y aristas.
Dado que la estructura es genérica, no es posible implementar funciones que requieran especificar al menos una parte de la misma, por lo cual se decide brindar funciones que permitan recorrer la estructura (desde el enfoque decidido por el usuario), obteniendo en cada nodo o arista recorrido toda la información disponible en cada caso.
Se decide mantener además una lista de listas de Objetos del Mapa (a cada una de estas listas le llamaremos layer), de modo de mantener unidos (a través de estas listas) los elementos que fueron ingresados en la misma capa temática.

Contexto de Trabajo (Software, Hardware, Sistema Operativo, etc.).

El lenguaje utilizado será C++, con lo cual puede ser utilizado en cualquier Hardware o Sistema Operativo que cuente con un compilador para C++, realizando un mínimo de cambios al código.

Formación ofrecida al estudiante.

Formación en el área de VRP y Sistemas de Información Geográfica (SIG -o GIS-).

Plan de Trabajo

Revisión bibliográfica sobre VRP, Estructura de Datos y SIG
Definición de requerimientos.
Análisis de las funciones necesarias
Diseño de las mismas
Desarrollo de interfase entre shapefiles y archivos .dbf con archivos de texto.
Implementación y testeo

Resultados esperados

El resultado del taller será una Estructura de Datos que ofrezca la posibilidad de resolver cualquier VRP

Interés académico

El taller esta "bien ubicado" en el tiempo dado la casi no existencia de este tipo de Estructuras. En particular, es de interés para el Dpto. de Investigación Operativa.

CONCLUSION Y RESULTADOS

Se cumplió con los objetivos planteados, implementando una Estructura de Datos Genérica que soporta los algoritmos de solución del Problema de Ruteo de Vehículos. Se puede apreciar a través de los resultados del Testeo que tanto levantar la estructura como los algoritmos que corren sobre ella funcionan en **tiempos muy aceptables**.
Se confirmó que C++ es un lenguaje adecuado para cumplir con los objetivos planteados, tanto por su eficiencia y generalidad, como por su posibilidad de acceso a recursos de bajo nivel y de levantar archivos binarios como tales.
Se observa que la única forma de utilizar todas las potencialidades de la estructura se debe conocer el lenguaje de programación C++. En caso de desearse que la estructura sea utilizada por un usuario que desconozca C++ habría que programar en C++ una interface con este usuario, perdiendo gran parte de la generalidad de la estructura.

ORGANIZACIÓN DEL INFORME

A continuación se explicará brevemente la organización del Informe.
El primer capítulo es una introducción al Problema de Ruteo de Vehículos.
El segundo, trata sobre algunos elementos del VRP .
En el capítulo 3 se trata sobre la estructura desarrollada para resolver el problema planteado, las razones que hacen necesario el desarrollo de la misma y las razones para escoger el lenguaje de programación para su implementación.
En el capítulo 4, se presenta los Requerimientos de la estructura de datos para problemas genéricos de Ruteo entregados por el representante de ICA (usuario).
En el capítulo 5 se presenta un diseño para la estructura, basado en los requerimientos y en las posteriores reuniones con el usuario.
En el capítulo 6 se presenta un plan de testeo genérico para un software desarrollado con los recursos con que se cuenta para un Proyecto Final, y luego, más en particular, un plan de testeo para la estructura desarrollada, resultados del mismo y tiempos de corridas.

El capítulo 7 presenta las conclusiones del actual trabajo, así como líneas por las que podría continuarse con el mismo, ya sea para ampliarlo o para mejorarlo.

Introducción al Problema de Ruteo de Vehículos

PROBLEMA

Comenzamos con este problema como forma de dar una base para la comprensión de la necesidad de contar con una Estructura de Datos genérica, que permita implementar algoritmos para la resolución de cualquier Problema de Ruteo de Vehículos.

Definición del problema

Introducción

El problema de ruteo consiste en obtener rutas que sean "buenas" (en algún sentido) para que recorran los vehículos visitando una sola vez cada punto de interés. Es decir que se debe decidir que recorrido hará cada vehículo (en general cumpliendo una función durante el mismo).
Los problemas de ruteo surgen fundamentalmente en el ámbito de la distribución/recolección tanto de bienes como de servicios. Éstos problemas implican trazar una o varias rutas a recorrer con determinadas restricciones, tratando de optimizar una determinada función objetivo.

Desarrollo

La logística puede ser definida como "la provisión de bienes y servicios de un punto de abastecimiento a un punto de demanda". Un sistema logístico completo, cubre todo el proceso de fabricación, es decir, desde el traslado de los insumos hacia las plantas, la conversión de los insumos en productos, el traslado de estos productos a depósitos y finalmente su envío a los consumidores finales.
Las actividades de *distribución* de una empresa comprenden todos los traslados y almacenaje de productos desde las plantas. El último paso de estos movimientos (de los centros de distribución a los clientes), que puede ser llamado transporte local o envío, es el eslabón más costoso de la cadena de distribución [1].
Para manejar efectivamente la distribución, se pueden distinguir tres niveles de toma de decisiones:

Estratégico
Ubicación de las facilidades (plantas y depósitos).

Táctico
Decisiones sobre el tamaño de los camiones (medios de transporte) y combinaciones posibles de los mismos.

Operacional
Ruteo y planificación de vehículos. Es en este nivel de enfoque de la resolución del problema que vamos a concentrarnos.

Hay que tener en cuenta que esta clasificación no puede tomarse muy rígidamente dada la cercana interacción existente entre las decisiones involucradas.
Por ejemplo, no es el mismo el problema de Ruteo si se decidió tener un solo depósito o varios, y a su vez uno o varios camiones, con la misma o distintas capacidades, que eventualmente podrían pasar por varios depósitos o estar asociados a un único depósito.
A pesar de esto, a partir de aquí, consideraremos únicamente los Problemas de Ruteo, como si los tres niveles de toma de decisiones fueran problemas independientes y con todas las otras decisiones ya tomadas.

La salida básica de todos los sistemas de ruteo y planificación es esencialmente la misma: Para cada vehículo se da una ***ruta*** *(sucesión de sitios a visitar) y una* ***planificación*** *(identifica las horas a las que las actividades en los sitios a visitar deben ser realizadas).*

Como una primera clasificación dividiremos los problemas en tres grupos: (a) ruteo, (b) planificación, (c) ruteo y planificación
Por ejemplo cuando no hay restricciones a priori sobre tiempos y si todas las mercaderías pueden ser enviadas en un período corto de tiempo (por ej. 3 horas). En este caso se puede decir que tenemos un problema de Ruteo de Vehículos puro. Si las horas de visitas a algunos sitios son importantes, entonces no puede ignorarse las características temporales y son éstas quienes guían las actividades de ruteo y planificación.

Definiciones

Un sistema de ruteo y planificación tiene un conjunto de entidades que requieren servicio. También existen un conjunto de restricciones temporales, entre las que podemos diferenciar:

- Una *relación de precedencia* entre dos de estas entidades indica que una de ellas debe ser servida antes que la otra.
- Otras restricciones son las impuestas por Ventanas de Tiempo (o *"time windows" en Inglés*). Por ejemplo: una ventana de dos lados [s,t] restringe el tiempo de servicio de una entidad a caer en el rango de horas comprendido entre s y t. Una ventana de un lado es de la forma $[-\infty, t]$, $[s, \infty]$ y lo restringe a caer respectivamente antes de t o después de s.

Si no hay restricciones de precedencia entre las entidades tenemos un **Problema de Ruteo Puro**. Si cada entidad tiene un tiempo de servicio, tenemos un **Problema de Planificación**. En otro caso, se tiene un **Problema de Ruteo y Planificación Combinado.** En general los Problemas de Ruteo y Planificación tienen tanto Relaciones de Precedencia como Ventanas de Tiempo.

EL PROBLEMA DE RUTEO DE VEHÍCULOS

Al resolver el Problema de Ruteo de Vehículos suele modelarse la Geografía como un Grafo con costos en las aristas y/o en los nodos. En estos términos estamos en condiciones de definir un problema básico de Ruteo de Vehículos. Tenemos un conjunto de nodos y/o arcos que deben ser servidos por una flota de vehículos. No hay restricciones sobre cuándo o en qué orden estas entidades deben ser servidas. El

problema es construir un conjunto factible de rutas de bajo costo (una para cada vehículo).

El Ruteo de Vehículos es primariamente un problema espacial. Se asume que no hay restricciones temporales, ni de otro tipo que afecten a la solución, excepto (tal vez) restricciones de máximo largo de la ruta. Este es, en contraste con los problemas de planificación [1], donde el movimiento de cada vehículo debe ser planificado en tiempo y espacio.

Complejidad

Problemas NP-Duros

Un algoritmo polinomialmente acotado es un procedimiento cuyo orden computacional crece polinomialmente con el tamaño del problema en el peor caso. La clase de los problemas para los cuales se conoce un algoritmo polinomialmente acotado se denota **P**. Los problemas que no pertenecen a P (no se conoce un algoritmo polinomialmente acotado para el problema) se llaman NP-Duros. Cuando uno se enfrenta a problemas ***NP-Duros***, se recurre frecuentemente a heurísticas o procedimientos aproximados para obtener soluciones cercanas a la óptima en lugar de buscar exactamente la óptima. Una algoritmo heurístico es un procedimiento que usa la estructura del problema de un modo matemático para obtener soluciones factibles o cercanas a la óptima. Muchas veces es posible obtener cotas para la diferencia con la solución óptima en el peor caso.

Problemas de ruteo

Los problemas de ruteo son NP-Duros [1], por lo cual interesa tener buenas heurísticas que los resuelvan (o encontrar un algoritmo polinomialmente acotado que lo resuelva) y es importante conocer la complejidad de los algoritmos a utilizar.
La mayoría de los problemas de ruteo y planificación pueden ser formulados como problemas de redes; una medida del tamaño del problema puede ser la cantidad de nodos (y arcos) de la red correspondiente.

Problemas de particiones y cubrimientos de conjuntos

Gran parte de los Problemas de Ruteo y Planificación pueden ser formulados como instancias de una clase especial de problemas de Programación Entera conocidos como Problemas de Particiones de Conjuntos y Problemas de Cubrimientos de Conjuntos. Básicamente un Problema de Cubrimiento de Conjuntos incluye una matriz de ceros y unos con costos asociados con las columnas. El objetivo es encontrar un conjunto de columnas de costo mínimo tal que el número de unos que aparecen el cada fila del conjunto seleccionado es por lo menos 1. Si el número es exactamente uno, entonces tenemos un Problema de Cubrimiento de Conjuntos. En la aplicación de estos problemas al Ruteo y Planificación de Vehículos las filas representan las entidades que requieren servicio y las columnas corresponden a formas en las cuales el servicio puede ser provisto a un subconjunto de las entidades demandantes. Aunque esto de un marco conceptual para la formulación de muchos problemas de ruteo y planificación, su utilidad práctica está limitada si el problema de programación entera resultante es muy grande. En este caso pueden utilizarse heurísticas para ello.

PROBLEMAS DE PLANIFICACIÓN DE VEHÍCULOS [2]

Estos problemas pueden ser pensados como Problemas de Ruteo con restricciones adicionales que tienen que ver con las horas a las que deben realizarse determinadas actividades. Los problemas de ruteo dan especial importancia a las características espaciales de las actividades realizadas, en cambio en los problemas de planificación a cada actividad se le asocia una hora, por lo cual los aspectos temporales de los movimientos del vehículo toman especial importancia.

En general la entrada de un Problema de Planificación de Vehículos es un conjunto de tareas. Cada tarea tiene especificadas una hora de comienzo, una hora de finalización, lugar de comienzo y lugar de fin. La función de costo consiste en componentes que pueden incluir costos de operación de vehículos. La flota de vehículos puede ser limitada y puede guardarse en uno o más depósitos. El tipo de problema resultante es una función de las restricciones impuestas sobre la formación de planificaciones, los tipos de tareas realizadas y los lugares donde dichas tareas deben ser realizadas.
Las restricciones del mundo real que habitualmente determinan la complejidad del problema de planificación de vehículos son las siguientes:
Cantidad de tiempo que un vehículo puede estar en servicio antes de regresar al depósito para service o cargar combustible.
Ciertas tareas que solo pueden ser realizadas por cierto tipo de vehículo.
La presencia de una variedad de depósitos donde los vehículos pueden ser guardados.

Problemas de Ruteo y Planificación Combinados

La mayoría de los Problemas de Ruteo y Planificación combinados se presentan como aplicaciones y están caracterizados por la precedencia de tareas y las ventanas de tiempo.
Por ejemplo, en el caso de la distribución, se fuerza una relación de precedencia entre la tarea de recoger algo y la de entregarlo, así como la restricción de que sean realizadas por el mismo vehículo, o que haya un intercambio entre la carga de dos vehículos, para lo cual deben coincidir en tiempo y espacio durante el tiempo necesario para la carga y descarga.

IMPLEMENTACIÓN

La implementación de sistemas para Ruteo de Vehículos incluye varias áreas de trabajo, que incluyen entre otras la recopilación de información del problema concreto, la elección de una estructura para representar el mapa, que está muy vinculada en general al algoritmo elegido (dado que en general se carece de una estructura genérica se busca implementar la que mejor se adecue al problema en particular y a los algoritmos a utilizar), la elección de una plataforma de trabajo, así como de un lenguaje de y un estilo programación.

[2] También se puede hablar de problemas de planificación de personal, cuyo tratamiento es bastante parecido al de planificación de vehículos.

Datos requeridos

Distancias y/o tiempos de viaje
Costos
Requerimientos del servicio

En general estos problemas tienen requerimientos de las tres categorías, si bien en muchos casos, para su solución se fija un modelo con una consideración primaria de minimizar una de estas categorías. Tanto las restricciones como los datos pueden ser simplificaciones de los reales, para permitir que el modelo sea aplicable a problemas reales.

La primer categoría, la distancia es probablemente la más fácil de manejar. Se dispone de información sobre distancias entre dos ciudades o pueden realizarse cálculos sobre un sistema de coordenadas. Esto se complica más cuando se requiere el tiempo de viaje pues el mismo depende de la hora del día y la zona del viaje.
En las otras dos categorías puede ser mucho más difícil la obtención de los datos. Los costos de operación de los vehículos pueden ser difíciles de estimar con precisión. La demanda del servicio puede no estar determinada (por ejemplo en la recolección de basura) por la naturaleza estocástica del problema.

Sistemas de computación interactivos

Uno de los mayores impedimentos para la aceptación de los sistemas de computación es un sentimiento de pérdida de control sobre el sistema físico subyacente. Muchas soluciones generadas por la computadora son rechazadas por problemas relativamente menores que podrían ser corregidos si algunos controles sobre el sistema se dejaran para el usuario. Incluyendo la interacción dinámica hombre-computadora se han logrado mejores soluciones y una mucho mayor aceptación por parte de los usuarios.
Dos grandes contribuciones que el usuario puede hacer dinámicamente son:

1. Incluir restricciones implícitas
   Una ventaja de esto es que no es necesario incluir todas las restricciones desde el principio sino que puede incluirlas el usuario en caso que la solución que se está tratando de determinar viole la restricción. Esto permite simplificar el modelo, dado que en muchos casos es probable que la restricción implícita no sea violada por la solución, así como también facilita al usuario la identificación de restricciones que si bien conoce intuitivamente, no identifica hasta que las ve violadas.

2. Usar la intuición humana para ayudar a un algoritmo heurístico.
   Hay muchas situaciones en que la intuición humana no puede ser fácilmente programada en un algoritmo heurístico.

Consideraciones de confiabilidad

Todos los sistemas humanos o físicos experimentan fallas. El diseñador del sistema debe decidir en la fase de modelado cuan explícitamente tomar en cuenta las consideraciones de confiabilidad. En un extremo, se puede incluir la confiablidad del sistema en la función objetivo o el conjunto de restricciones. Alternativamente, se puede

hallar una solución sin tener en cuenta las fallas de los componentes del sistema y luego evaluar la falla de ciertos componentes del sistema y realizar planes de contingencia.

Fallas más significativas:

- Vehículos – Por ejemplo roturas no planificadas
- Personal – Por ejemplo enfermedad
- Red utilizada por el sistema – Por ejemplo mal tiempo que inutilice algunos arcos (o nodos).

El sistema puede hacerse más confiable teniendo recursos de reserva.
En el caso de vehículos o personal la solución consiste en tener vehículos y personal de reserva; esto también presenta problemas, por ejemplo si la red en muy grande, la falla de un vehículo en un sitio no puede ser remediada a corto plazo por otro vehículo que está lejos de dicho sitio.
En el caso de fallas de la red, por ejemplo mal tiempo en un aeropuerto debe haber uno de emergencia no muy lejos y combustible suficiente como para llegar allí (lo cual hace más fuerte la restricción de longitud máxima del viaje).

Bases de datos para ruteo y planificación

En el caso de paquetes de software para resolver estos problemas resuelve repetidamente el problema de ruteo y planificación para un sistema particular de distribución, se requiere tener facilidades para el manejo de datos flexibles y orientadas al usuario.

Elementos del Problema de Ruteo de Vehículos[2]

INTRODUCCIÓN AL PROBLEMA DE RUTEO DE VEHÍCULOS

Definición del problema

El Problema de Ruteo de Vehículos (VRP) ha sido una de las más populares aplicaciones de la investigación de operaciones de la última década. La mayoría de los observadores atribuyen esta popularidad a la gran relación que existe entre la teoría y la práctica en el VRP, otros la atribuyen a la gran cantidad de dinero que se invierte anualmente en transporte de personas y mercaderías, de ahí que se busque una solución óptima para este problema.
El mayor logro en el VRP ha sido, poder capturar la gran mayoría de las características del "mundo real", haciendo que sus respuestas sean útiles y por sobre todas las cosas aplicables a la realidad.

Información relevante

Antes de comenzar a trabajar en el problema, debemos identificar a que tipo de problema nos estamos enfrentando.
A continuación veremos algunos ejemplos del tipo de información que debería recabarse.

- Capacidad del medio de transporte utilizado, por ejemplo si el problema se tratase de una compañía de buses escolares, nos estaríamos refiriendo a la cantidad de asientos que tiene cada bus.
- Cantidad de conductores.
- Número de rutas que se recorren diariamente, y la cantidad de paradas que existen en cada ruta.
- Operaciones inter-ciudad o intra-ciudad.
- Costo anual de las actividades de distribución.
- Identificación del porcentaje del costo funcionamiento que corresponde a actividades de distribución.
- Estimaciones de la demanda y área de servicio futuras.
- Cartografía de la zona sobre la que se opera o se espera operar en el futuro.
- Capacidades actuales de computación y soporte para ruteo.
- Integración del ruteo con otras actividades
- Si se pretende que el sistema de ruteo se integre a los sistemas ya existentes de la empresa, o que por le contrario el sistema de ruteo sea totalmente independiente del resto de los sistemas informáticos de la empresa.

Esta información da una idea inicial de la magnitud de los recursos que la empresa puede asignar a la distribución. Con ella, el experto puede calcular el rango de costos del sistema que la empresa puede justificar económicamente.

Las macro características de la actividad de distribución y la práctica diaria de la empresa, pueden ser tomadas como guías generales para estimar los beneficios potenciales de instalar un sistema de ruteo o de despacho más elaborado.

Clases de problemas de Ruteo de Vehículos

Los problemas de Ruteo de Vehículos generalmente caen en una de estas tres categorías.

1) Solo de recolección, o solo de entrega.
2) Solo de recolección, o solo de entrega, pero con la posibilidad de entrega diferida.
3) Combinando recolección y entrega.

A la hora de enumerar los tipos de problemas de Ruteo de Vehículos estos no son los únicos tipos que existe ya que si tomamos en cuenta nuevas restricciones encontraremos una clasificación aún más fina de los mismos, en donde tenemos para cada una de los tipos anteriores:

1) Problemas en donde solamente una selección del conjunto total de clientes deben ser atendidos.
2) Problemas donde se desconoce el tamaño de los paquetes a ser entregados, o recolectados.
3) Problemas en donde existe la posibilidad de satisfacer la demanda de un cliente en distintas etapas.
4) Problemas en donde se debe tomar en cuenta que la entrega o recepción en los distintos clientes debe realizarse dentro de un rango horario preestablecido.

Muchas aplicaciones prácticas de los problemas que enumeramos aquí pueden encontrarse a diario en la vida, real de ahí como mencionamos anteriormente el gran desarrollo del área de VRP en la última década.

Características operacionales de las rutas

Una vez que las características del problema han sido determinadas, se deben estudiar de la forma en que actualmente se determinan las rutas dentro de la organización, de esta forma se establecen restricciones adicionales a las que surgen del problema en sí. A modo de ejemplo podemos considerar el caso en donde el último destino de un vehículo puede ser la casa del conductor del mismo y no el depósito.

Todas estas características si bien no aparecen como parte en si del problema, deben ser tomadas en cuenta a la hora de establecer “la mejor solución” al problema.

Adicionalmente a todas estas características en algunos problemas debemos tomar en cuenta la capacidad de los vehículos y temas relacionados con el proceso de carga de los vehículos, por ejemplo en un sistema de reparto de víveres, aquellos que deben ser entregados al último cliente en la ruta deben ser cargados primero; de esta forma el orden en que se visiten los clientes junto con la capacidad de los camiones se han convertido en variables de decisión importantes en el problema.

Otro tipo de características que condicionan el problema nacen cuando estamos frente a problemas de ruteo que toman características del tránsito de vehicular en la ciudad, en donde será de gran interés el evitar en nuestras rutas, vías de tránsito congestionadas, giros a la izquierda, giros en U, y otro tipo de maniobras que pueden considerarse como riesgosas en el tránsito.

RESOLVIENDO EL PROBLEMA DE RUTEO

Formulación del modelo y objetivos del mismo

Una vez que se ha determinado el tipo de problema y las restricciones que gobiernan las posibles rutas, se debe plantear un modelo que nos permita encontrar la solución. Para la formulación de este modelo se deberá poner especial énfasis en la búsqueda de la función objetivo que más se acerque al objetivo del problema real, es decir buscar enfatizar con ésta función objetivo los aspectos más relevantes del problema de ruteo al cuál nos enfrentamos.

Estrategias para llevar a cabo la implementación de la solución

Una vez que el problema ha sido establecido y se ha encontrado la función objetivo más apropiada a el problema debemos delinear un plan de trabajo que nos permita llevar a cabo nuestra solución, para ello como primer paso debemos determinar si algún paquete de software ya existente en el mercado se ajusta a nuestras necesidades o se requiere de una implementación especifica. Este estudio podrá ahorrar, y seguramente lo hará, mucho dinero y tiempo.

Técnicas algorítmicas

Hasta el momento para buscar nuevas soluciones a los problemas de ruteo han surgido tres técnicas, las mismas son:

a) El diseño de algoritmos con el objetivo de enriquecer las soluciones heurísticas ya existentes, esta técnica tiene la principal ventaja de permitir al desarrollador mantener la estructura original de la heurística y simplemente construir sobre la misma.
b) Utilización de Programación Matemática.
c) “Heurísticas Secuenciales”, se ha denominado así a la técnica que divide el problema, en una secuencia de subproblemas, en donde cada uno será resuelto en forma secuencial con diferentes algoritmos.

En todos estos casos vale la pena recalcar que la utilización de un diseño modular, contribuye a una mayor reutilización de código y funciones, hecho este que conlleva a un menor trabajo de programación y por lo tanto a menor esfuerzo, recordemos que es en vano “reinventar la rueda”.

IMPLEMENTACIÓN DE SISTEMAS DE RUTEO DE VEHÍCULOS

A la hora de llevar a cabo la implementación de un sistema de Ruteo de Vehículos debemos establecer de que forma el sistema interactuará con los usuarios. Será a partir de esto último que podremos definir la respuesta que el sistema deberá devolver al usuario; por ejemplo si todas las decisiones de ruteo deben ser tomadas por un despachador, el sistema deberá implementar algún algoritmo de ruteo en base a las decisiones tomadas por el despachador, otro tipo de interacción puede darse en caso de

que la toma de decisiones la haga el propio sistema, y el despachador tenga como única tarea la aprobación o reprobación de las rutas sugeridas por el sistema.
Otra importante consideración que deberá tomarse en cuenta es el grado de integración que deberá tener con el resto de los sistemas informáticos de la empresa, esto se hace especialmente interesante en aquellas empresas en donde existe un sistema de información geográfica.

Desde el punto de vista de una implementación en concreto sería bueno el mantener índices y estadísticas que nos permitieran evaluar el desempeño del sistema de ruteo implementado y los beneficios que trajo este a la empresa. De esta forma pueden corregirse carencias del sistema buscando una mayor eficiencia, lo cual facilitará un mantenimiento correctivo del sistema.

SISTEMAS DE INFORMACIÓN GEOGRÁFICA

Podemos decir que un Sistema de Información Geográfica (GIS por sus siglas en Inglés) es un sistema informático capaz de reunir, almacenar, manipular y visualizar información geográficamente referenciada, es decir datos identificados por su ubicación en un mapa[3].
La incorporación de Sistemas de Información Geográfica, han traído grandes beneficios al área de VRP, facilitando la visualización, tanto de los problemas como de las soluciones a los mismos.

La información básica requerida de un sistema de información geográfica es la siguiente:
a) Localización precisa de los clientes.
b) Distancia entre los distintos puntos del mapa.
c) Tiempos de viaje entre los distintos puntos del mapa.
d) Información relevante a las distintas rutas del sistema, como puede ser intensidad de tráfico por las distintas calles de nuestra ciudad.

3 Ver Apéndice 3

Estructura

¿POR QUÉ HACER LA ESTRUCTURA?

Como ya se ha mencionado en capítulos anteriores se carece de una estructura general sobre la que desarrollar los algoritmos para resolver cada caso particular. Esto conlleva un costo en tiempo de programación al tener que manejar toda la representación geográfica, en todos los casos. Tener una estructura genérica permite olvidar toda la parte de representación geográfica y concentrarse en la optimalidad del algoritmo. Es evidente que la estructura debe funcionar eficientemente, dado que en caso contrario, puede no ser útil por enlentecer hasta tiempos inaceptables los algoritmos implementados sobre la misma.

LENGUAJE DE PROGRAMACIÓN

Introducción

Para elegir el lenguaje de programación a utilizar se toman en cuenta los requerimientos del problema, así como también de la propuesta de Taller.
Nuestro taller debe estar programado en un lenguaje de programación no orientado a información geográfica, de modo que permita a la empresa comparar resultados con lo realizado por otro grupo que implementa lo mismo en un lenguaje orientado a información geográfica. La propuesta original era programar en C, teniendo en cuenta sus características de acceso a bajo nivel, eficiencia y multiplataforma. También hay que tener en cuenta que por tratarse de un Proyecto de fin de carrera, el mismo debe terminar en a lo sumo un año, por lo que, en caso de considerar como más apropiado un lenguaje que nos resulte desconocido, hay que evaluar también los costos (en tiempo) de aprender un lenguaje nuevo.

¿Por qué C++?

En la propuesta del Taller se plantea el interés en contar con una estructura de datos genérica para problemas de ruteo, cuya implementación se sugiere utilizar el Lenguaje de Programación C. En todo momento se hace especial énfasis en la generalidad de la estructura de forma tal que sobre la misma puedan correr una cantidad importante de algoritmos de ruteo. Esta característica de generalidad, coincide plenamente con los objetivos que busca la herencia en los lenguajes orientados a objetos. Por otro lado vemos que C++, es un sucesor directo de C, el cuál incorpora a su antecesor el concepto de clase, podemos afirmar a grosso modo que C++ es una extensión del lenguaje C con orientación a objetos.

Uno de los paradigmas bajo los cuales fue concebida la programación orientada a objetos es principalmente la generalidad y reutilización de código, creando clases que respondan a objetos básicos de nuestra realidad, esta característica inherente a la teoría de programación orientada a objetos hace que la herramienta más adecuada para la implementación de una estructura con las características de esta sea un lenguaje orientado a objetos. En nuestro caso particular se sugería que el lenguaje de

programación a utilizar fuera C. Combinando las conclusiones anteriores surge que C++, resulta la herramienta adecuada para desarrollar una estructura de datos genérica para la resolución de problemas de ruteo. Esta decisión es además óptima en tiempo de desarrollo del sistema, puesto que C++ es uno de los lenguajes de programación orientada a objetos que conocemos mejor.

Análisis de Requerimientos

INTRODUCCIÓN

A partir de la solicitud planteada por el Área de Optimización de la empresa ICA [3], y de conversaciones posteriores con su representante para este proyecto, Ing. José Comas, se elabora este capítulo, destinado a analizar y especificar dichos requerimientos, de modo que permita tomar este capítulo como base de la estructura a implementar, una vez acordado con la empresa. Sólo se tomará en cuenta la parte de la especificación de requerimientos que afecte a la estructura implementada en C++ que es la correspondiente a este proyecto.
En el apéndice 1 se presenta el documento original de requerimientos entregado por el representante de la Empresa. Este capítulo puede considerarse una versión posterior del mismo. En él se resumen los requerimientos originales de la Empresa, así como nuevos agregados a los mismos y comentarios sobre las partes originalmente solicitadas que finalmente se acordó no implementar.

PROBLEMÁTICA

Los Problemas de Ruteo tienen dos componentes importantes, la Estructura de Datos sobre la cual se corren los algoritmos, y los algoritmos en sí. En nuestro caso la preocupación es la estructura de datos para los algoritmos. Cada algoritmo en particular tiene asociada una estructura que se adecua al mismo de la mejor manera, pero cualquier cambio en dicho algoritmo hace que la estructura quede obsoleta o que haya que volver a diseñarla, por lo que es importante una buena estructura de datos que soporte una gran grama de problemas. En particular, debe soportar todos los Problemas de Ruteo de Vehículos, pero también algunos otros problemas asociados como por ejemplo los Problemas de Asignación de clientes a depósitos.

OBJETIVO FINAL

Como objetivo del Taller V se busca tener una estructura de datos con sus correspondientes funciones de manejo, que permite montar sobre la misma todos los componentes de un problema de “Ruteo de Vehículos”, resolverlo y recuperar los resultados.

CARACTERÍSTICAS DEL USUARIO FINAL

En el caso de este proyecto, el usuario final, es un programador C++, que está capacitado para realizar modificaciones y mejoras a la estructura, en caso de ser necesario para su aplicación a casos particulares. A pesar de esto, el usuario desea poder usar la estructura, sin tener que realizar modificaciones a la misma, por lo cual debe estar debidamente documentado en el Manual de Usuario que se adjunta, de qué modo pueden usarse los objetos de la estructura, sin tener que modificar su implementación.
Se acuerda con el usuario que los errores en la llamada a funciones y/o en el pasaje de parámetros son de su entera responsabilidad, por lo cual, los controles internos que

realice la estructura a estos efectos son voluntarios por parte de sus implementadores y sólo a los efectos de tener una estructura más robusta.

ENTRADA DE DATOS

Los datos se pueden dividir en dos categorías, los datos geográficos y no geográficos. Se tomará esta clasificación como base, puesto que permite a su vez distinguir entre los datos que serán de mayor interés para la empresa (geográficos) y los que no lo serán en la misma medida (no geográficos).

Tipos de archivos a utilizar

El programa Arcview (que es un GIS [16]) guarda la información en dos archivos: con los siguientes formatos:

- Shapefile – Formato que utiliza el Arcview para manejar la información geográfica y que tiene un formato fijo [4]
- Dbf - Formato de la base de datos dbase III y posteriores que es legible o importable desde la mayoría de los manejadores de bases de datos y que tiene un formato fijo[14]. Estos archivos son utilizados por el Arcview como forma de dar información adicional a la solamente geográfica del shapefile.

Datos Geográficos

Son aquellos que tienen una componente geográfica. Esto es, que tienen coordenadas en algún sistema de referencia y que este dato es muy importante para su uso posterior.
Están dados en formato shapefile de: líneas y puntos (el shapefile de polígonos no es un dato de entrada).

Shapefile de Líneas:

Los shapefile de líneas corresponden a la red sobre la cual se realizan los cálculos de ruteo. Debemos poder cargar varios shapefile de líneas uno sobre los otros, por ejemplo: un shapefile de calles, otro de avenidas, otro de puentes, etc. y que la unión de éstos forme la red sobre la cual vamos a colocar los puntos de interés (depósitos, clientes, etc.).

En el archivo **.dbf* de cada shapefile, existe un conjunto de datos que deben ser tenidos en cuenta por la estructura de datos, estos son:

- *Id:* Identificador del usuario, es un entero.
- *Largo:* largo del arco, es un real. (No Se Calcula, es solo un costo más al que se desea llamar largo)
- *Tiempo:* tiempo de viaje del arco, es un real.
- *Flecha:* indica el flechamiento de las calles, este campo puede tener tres valores:
  0: cuando la calle no esta flechada
  1: cuando el flechamiento es en el sentido de la digitalización
  -1: cuando el flechamiento es en sentido contrario a la digitalización.
  Si se omite en algún caso este dato (pero se están ingresando flechas), se asume que la calle no está flechada (0)
- *Habilitado:* indica si se encuentra habilitado dicho arco:

0: no esta habilitado
1: está habilitado.
Por defecto está habilitado

Además se tendrán varios pesos o impedancias, que serán utilizados como costos y varias "Flags", que servirán para tomar en cuenta o no una determinada línea en el correr de los algoritmos.

Estos campos no tienen porque encontrarse en los datos, ya que por ejemplo si la red no es flechada, el campo *flecha* es superfluo. Lo mismo sucede con el *Id*, éste es un identificador del usuario por lo que puede encontrarse o no, además de encontrarse no se asegura que los datos sean confiables ya que el manejo de los mismos lo hace el usuario y no la estructura.
El campo *Habilitado* corresponde al conjunto de las "flags" que puede tener un arco.

Datos adicionales al shapefile de líneas

Hay otras consideraciones que vamos a realizar, en este caso especificado anteriormente existen un par de campos que indican el "peso" del arco, éstos son *Largo* y *Tiempo,* de todas formas puede haber un conjunto mucho mayor de "pesos" en cada arco, por ejemplo: varios tiempos indicando velocidades distintas de viaje según horas del día o días de la semana. Para estandarizar los nombres de los campos y sus datos, junto con el shapefile de líneas, tienen como entrada un archivo ASCII que indica que campo tiene que información y cuál es el tipo de datos de dicha información. Este archivo se llama como el shapefile pero con extensión *inf.*

El formato del archivo es el siguiente :

[IdUsuario]
Nombre del campo de ID usario , Tipo de datos: este puede ser vacío
[Flecha]
Nombre del campo de flechamiento , Tipo de datos: puede ser vacío.
[Habilitado]
Nombre del campo de habilitacion , Tipo de datos: puede ser vacío.
[Largo]
Nombre del campo largo, Tipo de datos: puede ser vacío.
[Tiempo]
Nombre del campo de tiempo, Tipo de datos: pude ser vacio
[peso1]
Nombre del campo de peso1, Tipo de datos
[peso2]
Nombre del campo de peso2, Tipo de datos
...
...
[pesoN]
.....
[Flag1]
Nombre del campo de la flag1, Tipo de datos
[Flag2]
Nombre del campo de la flag1, Tipo de datos

.....
[FlagM]

Al menos uno de los campos que corresponde al "peso", o sea: *largo, tiempo, peso1..peson*, tiene datos, porque de lo contrario los algoritmos de Ruteo no tiene un costo de transporte.
La estructura conservará el índice de cada "peso" y de cada "flag", junto con su nombre.

Los tipos de datos que debe soportar cada campo descripto anteriormente es el siguiente:

[IdUsuario]	entero
[Flecha]	entero
[Habilitado]	entero(igual que una flag)
[Largo]	float/double
[Tiempo]	float/double
[peso1..n]	float/double
[Flag1..n]	entero

Shapefile de puntos

Los shapefile de puntos corresponden a los puntos de interés, como por ejemplo depósitos, clientes, paradas obligatorias, etc.

De la misma forma que los shapefile de líneas tienen datos de importancia en el archivo *dbf*, los shapefile de puntos tiene un conjunto de datos de relevancia.

- *Id:* Identificador del usuario, es un entero.
- *Nombre*: Nombre del punto (por ejemplo del comercio)

Además se tendrán varias demandas (en el caso de clientes) o capacidades iniciales (en el caso de depósitos), y varias "Flags", que servirán para tomar en cuenta o no un determinado punto al correr de los algoritmos.

De la misma forma que en el shapefile de líneas, estos datos pueden o no encontrarse.

Datos adicionales al shapefile de puntos

Para estandarizar los nombres de los campos y sus datos, junto con el shapefile de puntos, tienen como entrada un archivo ASCII que indica que campo tiene que información, este archivo se llama como el shapefile pero con extensión *inf*.

Este archivo tiene el siguiente formato:

[IdUsuario]
Nombre del campo de ID usario: este puede ser vacío

[Demanda/Capacidad inicial 1]
Nombre del campo de demanda o capacidad: puede ser vacío.
[Demanda/Capacidad inicial 2]
.....
[Demanda/Capacidad inicial K]
.....
[Flag1]
Nombre del campo de la flag1
[Flag2]
Nombre del campo de la flag1
.....
[FlagM]
.....
[Nombre]
Nombre del campo "Nombre" del punto

Los tipos de datos que debe soportar cada campo descripto anteriormente es el siguiente:

[IdUsuario]	entero
[Demanda/Capacidad inicial 1..k]	float
[Flag1..m]	entero
[Nombre]	string

Datos No Geográficos

Los datos que no son geográficos son aquello que no tienen una componente geográfica tales como las reglas de tránsito, los vehículos, etc.

Todos estos datos están dados en archivos ASCII, con cabezal y delimitados por comas de nombres. Eventualmente podrán ser archivos .dbf para asemejarse al resto de los datos:

- *Reglas.txt*: reglas de tránsito
- *Vehículos.txt:* los vehículos
- *Semáforos.txt:* semáforos

La estructura modela y toma en cuenta estos datos para todas sus funcionalidades y algoritmos, pero no provee métodos que permitan la carga automática de dichos archivos.

Reglas de tránsito

Indicar las reglas de tránsito, o sea, prohibiciones de doblar, costo de hacer ciertas maniobras en una esquina, realizar un giro, etc. Estos datos están en un archivo ASCII de nombre *Reglas.txt* y en el siguiente formato:

"Id_arco_origen","Id_arco_destino","costo"	cabezal
Id_arco_origen, Id_arco_destino, costo	datos de cada linea
.....	

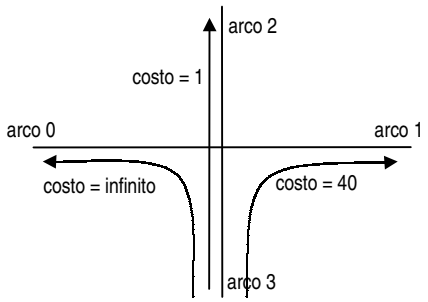
.....

el cabezal del archivo puede tener otros nombres, pero el orden en el cual se encuentran los campos no cambia. Se observa que el cabezal del se vuelve innecesario, pero se respeta la solicitud de la empresa a este respecto.

Los *id* de los arcos, son los usados por el usuario.

No tiene por que haber una regla de tránsito para cada esquina, todo lo contrario, lo normal es que algunos cruces de calles tengan reglas de tránsito.

Ejemplo del archivo:



Supongamos que tenemos esa boca calle, con los arcos, 0..3, y las prohibiciones que se indican con sus costos, el archivo tendrá la siguiente información:

Id_arco_origen,	*Id_arco_destino,*	*costo*
3,	2,	1
3,	0,	infinito (lo representamos con un numero grande)
3,	1,	40

Por defecto el costo de un maniobra es 0.

Semáforos

Los semáforos tienen una componente geográfica, su ubicación, pero en nuestros problemas no nos interesa donde se encuentras posicionados sino las demora que genera cruzar uno. Estos datos están en un archivo ASCII de nombre *Semaforos.txt* y tiene el mismos formato que el de *Reglas*, con la diferencia que el costo es el tiempo (o su equivalente en distancia) que se pierde por cruzar dicho semáforo en una dirección. Este "peso" que tienen los semáforos debe ser tomado en cuenta por las funciones que implementan sobre la estructura.

Vehículos

No tienen componentes geográficos, es decir que a los efectos de la estructura, no interesa su ubicación (ni siquiera en el instante inicial).
Los datos que pueden tener los vehículos son los siguientes:

- *Id*: identificador del usuario (entero)
- *Capacidad:* capacidad del vehículo (real), los vehículos pueden tener mas de un compartimento.
- *Twmin:* horario a partir del cual puede ser usado (entero - minutos)
- *Twmax:* horario a partir del cual es vehículo queda fuera se servicio. (entero – minutos)
- *Tiempo de Atención:* Tiempo que demora el vehículo en cumplir su función, una vez en la ubicación del cliente.
- *Nombre:* nombre del vehículo (string)

Como cada uno de estos campos puede o no encontrarse, además del archivo de vehículos, hay un archivo *vehiculos.inf* que contiene la descripción de los datos contenidos en *vehiculos.txt*.

> [IdUsuario]
> *Nombre del campo de ID usario*: este puede ser vacío
> [Capacidad 1]
> *Nombre del campo de capacidad del vehículo:* puede ser vacía, indicando capacidad infinita.
> [Capacidad 2]
> .....
> [Capacidad K]
> .....
> [TWmin]
> *Nombre del campo que indica el comienzo de la Ventana de Tiempo del vehículo*
> [TWmax]
> *Nombre del campo que indica el fin de la Ventana de Tiempo del vehículo*
> [TiempoAtención]
> *Nombre del campo con el tiempo de atención*
> [Flag1]
> *Nombre del campo de la flag1*
> [Flag2]
> *Nombre del campo de la flag1*
> .....
> [FlagM]
> .....
> [Nombre]
> *Nombre del campo con el "nombre" del vehículo*

Los tipos de datos que debe soportar cada campo descripto anteriormente es el siguiente:

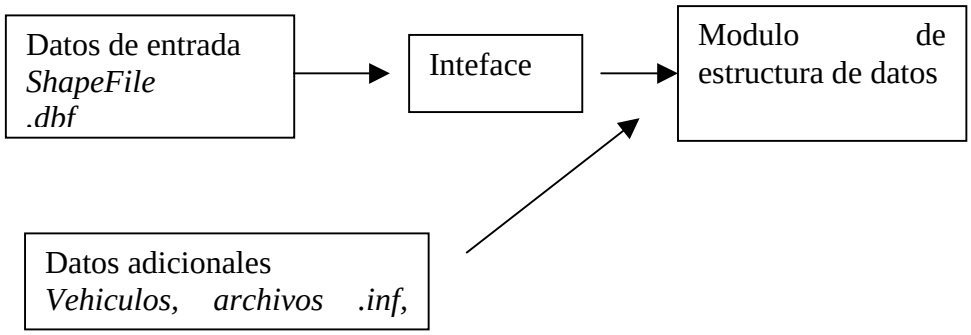
[IdUsuario]	Entero
[Capacidad1..k]	Float
[Twmin]	Float
[Twmax]	Float
[TiempoAtencion]	Float
[Flag1..m]	Entero
[Nombre]	String

Observaciones

Los nombres de los archivos de datos adicionales (vehículos, semáforos, etc.) no tienen por que ser los descriptos anteriormente, deben implementar las funciones necesarias para cargar esta información, teniendo como parámetro el nombre del archivo.

Estructura de datos

Se realiza una interface bien definida entre los shapefile y el módulo de la estructura de datos, de forma de independizarse del ArcView, y del formato shapefile.



Esta interface toma los shapefile y sus .dbf asociados y los transforma en un archivo ASCII delimitado por comas, el cual es la entrada a la estructura de datos.

SALIDA DE DATOS

La salida de datos se realiza en archivos de formato ASCII con cabezal delimitado por comas.
Los datos más importantes en una estructura de datos de este tipo es el estado de los objetos y en que orden deben ser recorridos, no debemos dejar de tener presente que la estructura de datos está implementada para soportar problemas de ruteo, retornando en forma clara la solución. Esta parte no se realizará

Objetos en la estructura

La solicitud de la empresa de que la estructura proporcione "listados de todos aquellos objetos con sus propiedades que se encuentran en la estructura" se cambia por la implementación de funciones que permitan recorrer la estructura, de modo que, una vez conocidos más detalles sobre la estructura (en particular los campos a listar) el usuario pueda fácilmente implementar las funciones de listado. Este cambio se debe a que al ser una estructura abierta, es imposible conocer de antemano cuáles van a ser los campos a listar.

FUNCIONALIDADES

Sobre los ShapeFile de líneas (RED)

Se puede cargar varios shapefile de líneas (redes) uno sobre los otros, por ejemplo: un shapefile de calles, otro de avenidas, otro de puentes, etc. y que la unión de éstos forme la red sobre la cual vamos a colocar los puntos de interés (depósitos, clientes, etc.). Se guardará una distinción entre los que fueron ingresados en diferentes shapefiles, a la que se denominará Capa.
Cabe aclarar que cuando nos referimos a calles, en realidad estamos hablando de cuadras, es decir de los segmentos de calle que van de una intersección con otra calle a la siguiente consecutiva.

Atributos adicionales de una RED

- *Capa:* entero que indica el número de capa en la cual se cargó la red, el usuario indica que número de capa le quiere dar.
  Para cargar dicho número debe recorrerse la estructura utilizando las funciones que la misma provee a esos efectos y seteando uno a uno estos datos.

  Del conjunto de redes cargados en distintas capas, se puede habilitar o deshabilitar un subconjunto de las mismas para trabajar en ese momento. Se puede trabajar con la red cargada en cada capa por separado o en el conjunto que el usuario desea.

Deben permitir agregar atributos adicionales de forma fácil a la estructura de datos, dejando en claro las operaciones que se deben realizar para operar con los mismos.

Seteo de atributos de la RED

Para setear dichos atributos debe recorrerse la estructura utilizando las funciones que la misma provee a esos efectos y seteando uno a uno estos datos.

Atributos a setear:
- *Id*: permite modificar el id del usuario
- *Flechamiento*: permite cambiar, habilitar y deshabilitar el flechamiento de un arco, también habilitar y deshabilitar todo el flechamiento de la RED.

- *Flags:* todas las flags, incluso “habilitado” pueden ser seteadeas.
- *Pesos:* todos los “pesos” incluso “Largo y Tiempo” pueden ser seteados

Consultas en la RED

Para consultar dichos atributos debe recorrerse la estructura utilizando las funciones que la misma provee a esos efectos y consultando uno a uno estos datos.

Atributos a consultar:
- *Id* de un arco
- *Flechamiento* de un arco
- Si se encuentra habilitado o no el flechamiento de la RED
- *Flag***n** de un arco
- *Peso***n** de un arco

Sobre los ShapeFile de puntos

Se deberá poder cargar varios shapefile de puntos (depósitos, clientes, etc.) todos sobre las redes previamente cargadas. Se puede indicar (a través de la capa) en que red se va a cargar, o en que redes se van a cargar. Distintos shapefile de puntos pueden cargarse en redes de diferentes capas.

Los puntos se pueden encontrar en cualquier punto sobre la red, e incluso fuera de ella. En la mayoría de los casos los puntos se van a encontrar fuera de la red, debido a los errores de datos, de todas formas debemos de alguna forma aproximar los puntos que se encuentran fuera de la red, a la red. Esto se hará buscando las aristas de la red que estén “más cerca” en algún sentido a cada punto.
Hay que definir qué será para nosotros “más cerca”. Tal vez lo más intuitivo sea utilizar la distancia euclídea y la proyección ortogonal, si bien hay que recordar que lo que para nuestra estructura es el “largo” no será en general dicha distancia. Por otro lado, debemos recordar que en el caso de tener “polylines” de mas de 2 puntos, sólo se tomará en cuenta los extremos, con lo cual será mucho más inexacto el tema de las distancias y las proporciones. Los puntos tendrán asociada una arista y un desplazamiento sobre la misma que, si se utiliza la proyección ortogonal estará dado como la distancia de la proyección ortogonal del segmento al origen de la arista sobre el largo de la misma. Esto significa, que si, por ejemplo, el segmento AB mide 10 unidades, y la proyección del punto sobre el segmento cae a 3 unidades de A, el desplazamiento será 3/10.

Analizando junto al usuario, la asignación de puntos a las Aristas, se observa que en un caso como el de la Figura 1, en el que se tiene la representación de la curva E1abE2
como el segmento E1E2, El Punto P se encuentra aproximadamente en la mitad de la curva, y sin embargo, su proyección ortogonal sobre el segmento cae mucho mas cerca de E2 que de E1, con lo cual, el desplazamiento será bastante mayor que ½.

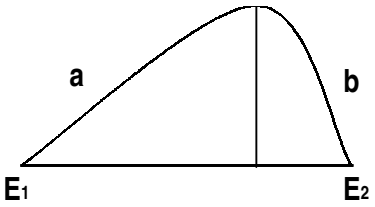


Figura 1.

Como solución a este problema se plantea la posibilidad de tomar en vez de la distancia antes mencionada para hallar el desplazamiento del punto sobre la arista la siguiente proporción: Se divide la distancia del punto al origen de la Arista entre la suma de la distancia al origen de la arista con la distancia a su destino. En la Figura 2 se muestra la proporción a utilizar, que sería, en ese caso: x/(x+y), que como puede observarse se aproxima mucho más a la proporción real sobre la curva. Además en caso que la curva coincida con el segmento determinado por el origen y el destino de la arista correspondiente, la proporción se parece a la de la proyección ortogonal en los casos en que el punto se encuentra próximo al segmento, lo cual es razonable en el caso de que la diferencia sea únicamente por errores en la toma de los datos.
A sugerencia del usuario, y para ser consistentes con la proporción utilizada, se acuerda tomar como distancia del punto a la Arista la suma de las distancias del punto al origen y al destino de la Arista.
Luego de implementada la estructura utilizando la medida sugerida por el usuario, presenta problemas en caso en que el punto se proyecte sobre una arista adyacente con otra proporcional mente mucho menor (sobre la que no se proyecta), pues con esta medida asignaríamos el punto a una arista sobre la que no se proyecta. Para minimizar los errores inducidos por este problema se cambió la distancia, utilizando para minimizar, en vez de la suma, la misma pero normalizada dividiendo entre la longitud de la arista (según lo visto en la Figura 2, sería (x+y)/c)

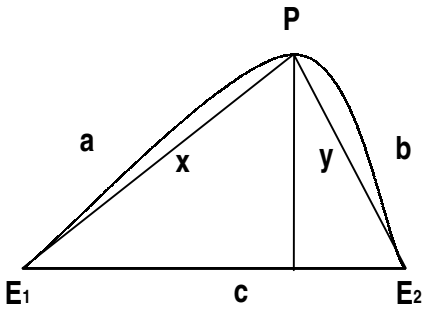


Figura 2.

En los requerimientos de la empresa se pide:
“El usuario puede dar un rango (por ejemplo distancia), a partir del cual los puntos que se encuentren a una distancia mayor que ésta dada, queden fuera de la red, o sea, no son alcanzables.” Se acuerda sin embargo omitir ese requerimiento, dado que los puntos que no estén en la red no interesan. Con la medida utilizada en este caso, todos los puntos serán asignados a alguna arista, si se utilizara la de la proyección ortogonal habría que definir además qué hacer con los puntos que no se proyecten sobre ninguna arista de la red.

Atributos adicionales de un punto

- *Capa:* entero que indica el número de capa en la cual se cargó, el usuario indica que número de capa le quiere dar.
- Algunos objetos como los clientes, pueden tener atributos adicionales tales como: *horaDeLlegada, horaDeAtención, horaDeSalida, CapaicadaODemandaRestantelN*
- *Visitado:* este pertenece a un conjunto de flags internas
- *Ruteado:* ídem a lo anterior.
- *Asignado:* ídem a lo anterior.

Deben permitir agregar atributos adicionales de forma fácil a la estructura de datos, dejando en claro las operaciones que se deben realizar para operar con los mismos.

Seteo de atributos de los Puntos

Para setear dichos atributos debe recorrerse la estructura utilizando las funciones que la misma provee a esos efectos y seteando uno a uno estos datos.

Atributos a setear:
- *Id*: permite modificar el id del usuario
- *Flags:* todas las flags, incluso "habilitado" pueden ser seteadeas.
- *Pesos:* todos los "pesos" incluso "Largo y Tiempo" pueden ser seteados
- *Capacidad***n:** cambiar las capaciadades
- *HoraDeLlegada*
- *HoraDeAtención*
- *HoraDeSalida*
- *Capacidad_Demanda_Restante***n:** cambiar la demanda o la capacidad N restante.

Consultas en los Puntos

Para consultar dichos atributos debe recorrerse la estructura utilizando las funciones que la misma provee a esos efectos y consultando uno a uno estos datos.

Atributos a consultar:
- *Id* de un arco
- *Flag***n** de un arco
- *Peso***n** de un arco
- *Capacidad***n:** consultar las capaciadades
- *HoraDeLlegada*
- *HoraDeAtención*
- *HoraDeSalida*
- *Capacidad_Demanda_Restante***n:** consultar la demanda o la capacidad N restante.
- Encontrar en una red particular todos los clientes que cumplen una Flag

Algoritmos

Se implementa un conjunto de algoritmos sobre la estructura de datos generada.
Cada algoritmo debe tomar como "peso" de la RED, el indicado por el usuario.

- Permite "navegar sobre la estructura", o sea, dado un objeto del tipo línea poder saber que conexión tiene con otros objetos y que costo le insume.
- Dado un punto permite encontrar: cual es el punto más cercano, lejano, o el que se encuentre dentro de un rango de costo particular; el par de puntos más próximos a él (hacia atrás y adelante), los n puntos más cercanos. Esto puede ser para un punto de la misma categoría, o de cualquier otra.

Algoritmos de prueba

Se implementa un conjunto de algoritmos para la prueba de la estructura de datos. Todos los algoritmos se resuelven usando pura y exclusivamente las estructuras de datos y la interface creada.

El algoritmo de Dijkstra que calcula la distancia mínima de un punto al resto. Este es necesario para poder tener la información de la distancia mínima entre dos puntos.

TSP

Introducción

El TSP (Traveling Salesman Problem – Siglas en Inglés por Problema del Vendedor Viajante) es un Problema de Ruteo de Vehículos con un solo depósito. Dicho problema requiere la determinación de un circuito de costo mínimo que pase por cada nodo en el grafo relevante exactamente una vez.
Si los costos son simétricos, esto es, si los costos de viajar entre dos ubicaciones no depende de la dirección del viaje, tenemos un problema del vendedor viajero simétrico; si no tenemos un problema asimétrico (o dirigido).
Toda solución factible para un problema del vendedor viajero simétrico se tiene dos arcos incidentes a cada nodo. En el caso del problema del vendedor viajero asimétrico hay un arco entrante y uno saliente a cada arco.

Modelización

Sea una red G = [N, A, C] siendo N el conjunto de nodos, A el conjunto de aristas y C la matriz de costos $C=(c_{IJ})$. Esto es, cij es el costo de moverse o la distancia del nodo i al nodo j. El problema del vendedor viajero requiere un circuito Hamiltoniano en G de costo total mínimo (un circuito Hamiltoniano es un circuito que pasa por cada nodo i en N exactamente una vez).

Características

Karp [1] ha probado que el TSP es NP-Completo. Esto significa que es improbable que existan algoritmos exactos polinomialmente acotados para el TSP. Algunos algoritmos

ingeniosos propuestos por Little et al [1]. tienen problemas con el almacenamiento y el tiempo de ejecución para más de 100 nodos. Una importante excepción es el reciente trabajo de Padberg y Hong [1].

Debido a la dificultad del TSP, se han desarrollado muchos procedimiento heurísticos (aproximados). Estas heurísticas pueden ser comparadas analíticamente, estudiando su comportamiento en peor caso. Alternativamente puede usarse la experimentación computacional para comparar la performance de estas heurísticas.
Presentamos a continuación el algoritmo de TSP de "Rosenkrantz, Sterns and Lewis", Nearest Insertion, que es el que solicita la Empresa que se implemente como prueba de la Estructura.

Algoritmo Tsp de "Rosenkrantz, Sterns and Lewis", Nearest Insertion (pag.88 – L.Bodin).

1. Comenzar con un subgrafo que consiste sólo en el nodo i
2. Encontrar el nodo k tal que c_{ik} es minimal y formar el subciclo i-k-i
3. *Paso de selección:* Dado un subciclo, encontrar un nodo k que no este en el subciclo más cercano a cualquier nodo del subciclo
4. *Paso de inserción:* Encontrar el arco (i,j) en el subciclo, que minimice $c_{ik} + c_{kj} - c_{ij}$. Insertar k entre i y j.
5. Volver al paso 3 a menos que se tenga un ciclo Hamiltoniano.

ASIGNACIÓN

Asignar clientes a depósitos, teniendo en cuenta la demanda de los clientes y las capacidades de los depósitos.

ASIGNACIÓN – TSP

Conectar el TSP anterior con la Asignación. Sobre cada asignación se realiza un TSP.

Descripción del Algoritmo de Asignación

Idea general

Para cada cliente no asignado, calcula la suma de: las distancias a cada depósito (aun no agotado, o sea que no excedió la capacidad) menos la distancia a su depósito mas cercano. Luego asigna aquel cliente que maximiza dicha suma, a su depósito mas cercano.

Especificación

Sea:
- C1.. Cn conjunto de clientes a ser asignados
- D1..Dm conjunto de depósitos
- K1..Km capacidad de cada depósito
- Dist(Cj, Dk) distancia del cliente j al depósito k

∀ Cj no asignado y Dk no saturado, se calcula: Dist(Cj, Dk)
sea Mj = Min(∀ k Dist(Cj, Dk)) o sea la distancia a su depósito más cercano.

Asigno aquel cliente tal que Max ((Σ Dist(Cj, Dk) en k) - Mj) a su depósito mas cercano.

El control de la capacidad de los depósitos que se realiza es la siguiente: primero asigno el cliente y luego controlo si el depósito se saturó, por lo tanto puede exceder la capacidad del depósito ya que si tiene una disponibilidad el depósitos de 10 y le asigno un cliente con 15, queda excedido el depósito en 5.

Luego de que un depósito se satura, se desasignan todos los clientes de los depósitos que aún no se encuentran saturados y se vuelve a comenzar con ellos.

Funciones

Aquí se especifican algunas de las funciones que se deben realizar sobre la estructura de datos. *Pueden cambiar los nombre y los parámetros de ser necesario siempre y cuando mantengan un criterio, además las funciones especificadas aquí no son las únicas, en caso de necesitar especificar algunas nuevas está librado a su buen criterio. Sugiero*

Sobre la red

Funciones que se aplican a la red (shapefile de líneas)

CargaRed(archivo)

Carga una capa de la red desde un archivo Ascii con el formato especificado en el Análisis.
Cada capa podrá contener arcos o locales. Una red puede ser cargada en sucesivas capas. En conjunto con la red, se especifican las impedancias de los arcos que forman la red así como demandas y capacidades de clientes y depósitos respectivamente.

Diseño

INTRODUCCIÓN

En este capítulo se lee el análisis buscando identificar y modelar los objetos de la realidad planeada en el mismo. El sistema solicitado por la empresa está destinado a un usuario programador, que conozca el lenguaje de programación. Debe enfocarse como una biblioteca que permita al usuario tener una representación de la estructura para poder implementar sobre ella los algoritmos de ruteo que necesite. Para la comprensión del mismo se sugiere leer primero los capítulos de Introducción al Ruteo de Vehículos, Los requerimientos planteados por la Empresa (Apéndice 1) y el Análisis de este sistema. La estructura Básica a modelar es un mapa y desde este punto de vista se enfoca el diseño a alto nivel.

DISEÑO

Realidad

La realidad de un mapa para ruteo plantea los siguientes objetos básicos:

- MAPA. Es la parte puramente geográfica.
  - Calles
  - Esquinas

- COSTOS. Son los objetos o situaciones (expresados en una unidad común) que presenten dificultades para la circulación sobre el Mapa.
  - Semáforos
  - Carteles de PARE
  - Giros
  - Distancias
  - Tiempo
  - Etc.
  -
- Objetos relevantes que se ubican sobre el mapa
  - Vehículos
  - Clientes
  - Depósitos

Modelización

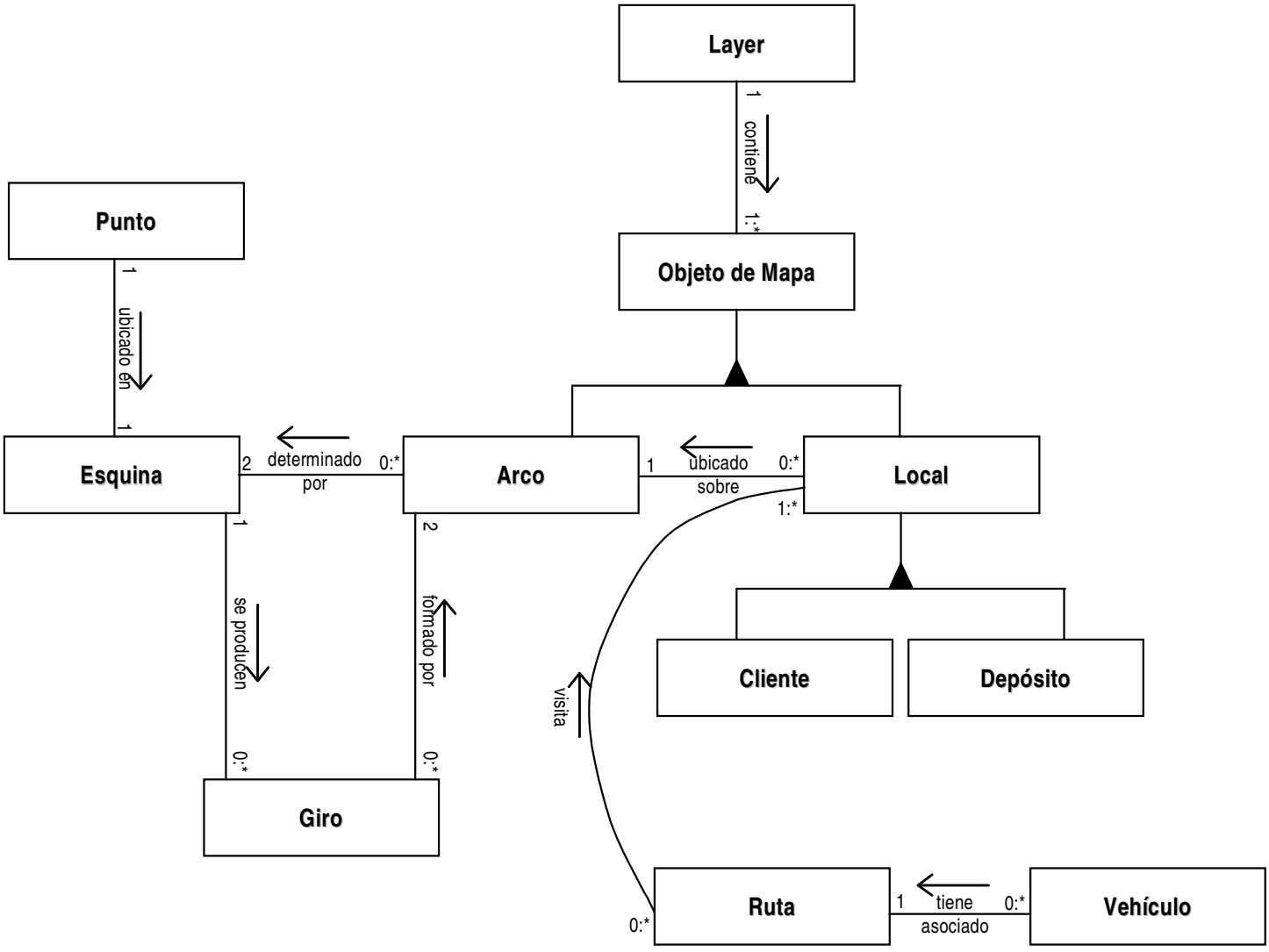
El mapa se modelará como un grafo dirigido (porque importa el sentido de circulación) A partir de los objetos mencionados y de la elección de un grafo dirigido para modelar el mapa se identifican los siguientes objetos en la realidad planteada:

1. Esquinas
2. Cuadras (Fracción de calle entre esquina y esquina)
3. Vehículos

4. Localidades a visitar (Depósitos y Clientes)
5. Giros (Opciones de doblar –o seguir derecho- que se presentan a un vehículo al llegar a una esquina)

Los objetos del 1. Al 5. tienen coordenadas asociadas en algún sistema dado. Dicha información permitirá hacer cálculos de distancias y mantener los datos para recuperar la geometría de dichos objetos.
Las esquinas son los nodos del grafo y las cuadras se representan con una arista (si la calle es de una sola mano) o por dos aristas (si la calle es doble mano). Tanto en el caso de los vehículos como en el de las localidades a visitar deberá asociárseles una (y una sola) cuadra. Dicha asignación se realizará buscando la cuadra cuya distancia al objeto sea mínima. Se fijará un rango de distancia en el que se considerará que el objeto está en la cuadra, si la distancia fuera mayor se lo considerará fuera del mapa.
Una esquina tiene (opcionalmente) asociado un conjunto de giros, cada uno de los cuales representará las posibles opciones de continuar nuestro recorrido pasando por la esquina. Se decide la inclusión de los giros para modelar el hecho de que el costo de pasar por una esquina dependerá no de la esquina propiamente dicha sino de la elección del camino a seguir en función del camino por el que se llegó. En casos en que los costos sean todos iguales (o no haya costos) no será necesario guardar todo el conjunto y no se hará para ahorrar memoria y tiempo de ejecución de los algoritmos de ruteo.
Habrá que pensar en una estructura similar para modelar el hecho de que un objeto se encuentre de un lado u otro de la calle. También en este caso la estructura deberá hacerse más compleja solo cuando sea necesario pues mantener la estructura más compleja puede requerir mucha más memoria y aumentar en mucho el tiempo que demoren los algoritmos.
Debemos recordar que el resultado de un Algoritmo de Ruteo es una ruta, por lo cual debemos también incluirlas en el modelo. Una ruta constará de un conjunto ordenado de cuadras. Como debemos conservar la información sobre la capa a que pertenece cada objeto, crearemos una lista de listas, cada una de las cuales une todos los elementos de la misma capa. A estos elementos genéricos les llamaremos Layers. De este modo, todo el mapa estará contenido en una lista de Layers.
También será importante tomar en cuenta que por un problema de precisión de los datos o de digitalización dos puntos que (en realidad) sean el mismo no estarán a distancia euclídea exactamente cero. Para resolver ese problema definiremos un valor (precisión) tal que consideraremos iguales dos puntos cualesquiera cuya distancia euclídea sea menor que ese valor.

MOR


Layer
1
contiene
1:*
Punto
Objeto de Mapa
1
ubicado en
1
Esquina
2
determinado por
0:*
Arco
1
ubicado sobre
0:*
Local
1:*
1
se producen
0:*
2
formado por
0:*
Cliente
Depósito
visita
Giro
0:*
Ruta
1
tiene asociado
0:*
Vehículo

TDN

Interface

Class **Interface**

uses **shapefile**
uses **dbf**
uses **txt**

exports

Procedure **Crear**();
// Crea un elemento de la clase interface

Function **Convertir_A_Texto**(s: **In Shapefile,** d: **In dbf**): **txt;**
// Toma un archivo de tipo shapefile y uno de tipo dbf y pasa la información a un archivo de texto con el formato que se detalla en [2]

Procedure **Eliminar** ()**;**
//Elimina un Elemento de la clase Interface

List

Generic Class **List(T)**

uses **T**
uses **Boolean**
uses **Entero**

exports

Procedure **Crear**();
// Crea una lista vacía

Function **Primero**():**T**;
// Devuelve el primer elemento de la lista.

Function **Ultimo**():**T**;
// Devuelve el último elemento de la lista.

Function **Siguiente**(Nodo: **In T**):**T**;
// Devuelve el último elemento de la lista.

Procedure **Insertar** (Nodo_a_Insertar: **In T**, Nodo_Ref **In T**, Modo: **In (Anterior, Posterior)**);
// Inserta el elemento Nodo_a_Insertar tomando como referencia para la inserción Nodo_Ref.. Si (Modo = Posterior) el objeto se inserta a continuación de Nodo_Ref. Si (Modo = Anterior) el objeto se inserta antes de Nodo_Ref.

Procedure **Eliminar_Nodo**(Nodo_a_Eliminar: **In T**);
// Elimina de la lista el elemento Nodo_a_Eliminar

Function **Cantidad_de_Elementos**(): **Entero**;
// Devuelve la cantidad de elementos de la lista.

Function **Vacia**():**Boolean**;
// Devuelve True si la lista contiene al menos 1 elemento y False en caso contrario.

Procedure **Eliminar** ();
//Elimina un List

Lista_De_Reales

Class **Lista_De_Reales** is **List(Real)**

Lista_De_Boolean

Class **Lista_De_Boolean** is **List(Boolean)**

Lista_De_Impedancias

Class **Lista_de_Impedancias** is **List(Impedancia)**

Nota: Impedancia puede ser cualquier clase que tenga definida una relación de Orden (es decir un operador < y un operador =)

Lista_De_Demandas

Class **Lista_de_Demandas** is **List(Demanda)**

Nota: Demanda puede ser cualquier clase Numérica

Punto_Cartesiano

Class **Punto_Cartesiano**

uses **Real**

exports

Procedure **Crear**(x,y: **in Real**);
// Crea un punto_cartesiano con abcisa x y ordenada y en un sistema cartesiano
Function **Su_Abcisa**():**Real;**
// Devuelve La Coordenada en el eje de las absisas de un sistema cartesiano
Function **Su_ordenada** ():**Real;**
// Devuelve La Coordenada en el eje de las ordenadas de un sistema cartesiano
Procedure **Cambio_Abcisa**(x: **In Real**);

// Asigna el valor x a La Coordenada en el eje de las absisas
Procedure **Cambio_ordenada** (y: **In Real);**
// Asigna el valor x a La Coordenada en el eje de las Ordenadas
Procedure **Eliminar** ()**;**
//Elimina un Punto_Cartesiano

Objeto_De_Mapa

class **Objeto_De_Mapa**

uses **Layer**
uses **Boolean**

Procedure **Crear**(L: **In Layer**, Enc, Exist: **In Boolean**);
//Crea un MapObject Que pertenece al Layer L, que esta encendido sii Enc es True y que existe lógicamente sii Exist es True

Function **Su_Layer**(): **Layer;**
// Devuelve el Layer al que pertenece (cada Objeto_De_Mapa pertenece a un y un solo Layer)
Function **Su_On**(): **Boolean**
// Devuelve True si el Objeto_De_Mapa esta prendido (activo) y False si no
Function **Su_Enabled**(): **Boolean**
// Devuelve True si el Objeto_De_Mapa Existe lógicamente y False si no
Procedure **Cambio_Layer**(L: **In Layer**)**;**
// Devuelve el Layer al que pertenece (cada Objeto_De_Mapa pertenece a un y un solo Layer)
Procedure **Cambio_On**(Enc: **In Boolean**)**;**
// Devuelve True si el Objeto_De_Mapa esta prendido (activo) y False si no
Procedure **Cambio_Enabled**(Exist: **In Boolean);**
// Si Exist True si el Objeto_De_Mapa Existe lógicamente y False si no
Procedure **Eliminar** ()**;**
//Elimina un Objeto_De_Mapa

Lista_De_Objetos_De_Mapa

Class **Lista_De_Objetos_De_Mapa** is **List(Objeto_De_Mapa)**

Layer

Class **Layer**

uses **boolean**
uses **entero**
uses **Info_Layer**
uses **Lista_De_Objetos_De_Mapa**

exports

Procedure **Crear** (**Tipo**:**in Boolean,** ID_Ext:**In Entero,** Enc: **In Boolean**, Inf**: In Info_Layer,** List_Obj: **In Lista_De_Objetos_De_Mapa**);
// Crea Un layer de puntos si Tipo es true y de líneas si tipo es false, El Identificador Externo será Id_Ext, el Layer esta prendido (activo) sii Enc = true, la Información adicional del layer será Inf y la lista de objetos del mismo será List_Obj

Function **De_Puntos**():**Boolean;**
//Devuelve true si es un layer de puntos y False si es un layer de líneas.
Function **Su_ID** ():**Entero;**
// Devuelve el Identificador interno del Layer
Function **Su_ID_Externo**():**Entero;**
// Devuelve el Identificador Externo del Layer
Function **Su_On**():**Boolean**;
// Indica si el Layer esta prendido (activo) o no
Function **Su_Informacion_Layer**():**Info_Layer;**
// Información adicional del layer cuyo tipo se instanciará después
Function **SU_Lista_De_Objetos**():**Lista_De_Objetos_De_Mapa**
Procedure **Cambiar_ID_Externo**(Id: **In Entero**);
// Devuelve el Identificador Externo del Layer
Procedure **Cambiar_On**(P:**In Boolean**);
// Si P es true marca el Layer como prendido (activo), si es false, marca el layer como apagado (inactivo)
Procedure **Cambiar_Informacion_Layer**(Info: **IN Info_Layer**);
// Cambia la información adicional del layer por Info
Procedure **Cambiar_Lista_De_Objetos**():**Lista_De_Objetos_De_Mapa**
// Cambia la lista de objetos del layer
Procedure **Eliminar** ();
//Elimina un Layer

Esquina

Class **Esquina**

uses **Punto_Cartesiano**
uses **List**
uses **Info_Esquina**
uses **Entero**
uses **Lista_De_ArcoS**
uses **Lista_De_Giros**

Exports

Procedure **Crear**(Ub: **in Punto_Cartesiano,** Inc : **In Lista_De_ArcoS,** Inf: **In Info_Esquina,**Gir: **In Lista_De_Giros**);
// Crea una esquina con ubicación (posición geográfica) ub, Lista de arcos incidentes Inc, Información de la esquina Inf y Lista de giros gir. La lista de Giros es la lista de costos de girar de un arco (incidente hacia adentro) a la esquina hacia otro (incidente hacia afuera de la misma esquina. Si un par de arcos no aparece en dicha lista se asume costo cero. Si es un giro imposible se asignará un costo infinito (la constante de la clase giro).

Function **Su_Ubicacion**():**Punto_Cartesiano;**
// Devuelve La ubicación de la esquina
Function **Sus_Incidentes** ():**Lista_De_Arcos;**
// Devuelve La Lista de arcos incidentes hacia fuera de la esquina
Function **Su_Informacion** ():**Info_Esquina;**
// Devuelve La información de la esquina
Function **Sus_Giros** ():**Lista_De_Giros;**
// Devuelve La Lista de giros de la esquina
Function **Su_Id** ()**:Entero**
// Devuelve el identificador interno de la esquina
Procedure **Cambio_Ubicacion**(U: **In Punto_Cartesiano**);
// Cambia la ubicación de la esquina por U
Procedure **Cambio_Incidentes** (Inc: **In Lista_De_Arcos**);
// Cambia La Lista de arcos incidentes hacia fuera de la esquina por Inc
Procedure **Cambio_Informacion** (Inf: **In Info_Esquina**);
// Cambia La información de la esquina por Inf
Procedure **Cambio_Giros** (G: **In Lista_De_Giros**);
// Cambia La Lista de giros de la esquina por G
Procedure **Eliminar** ();
//Elimina una esquina

Giro

class **Giro**

uses **Arco**
uses **Real**

Exports

Procedure **Crear**(Ori, Des: **In Arco,** Cost: **In Real);**
// Crea un Giro en que el Arco por la que se va es Ori, El Arco por la que se toma es Dest, y el costo de realizar este giro es cost
Function **Su_Origen**():**Arco;**
// Devuelve El Arco de origen del giro
Function **Su_Destino** ():**Arco;**
// Devuelve El Arco de destino del giro
Function **Su_Costo** ():**Real;**
// Devuelve el costo del giro
Function **Su_Infinito**():**Real;**
// Devuelve Valor de la constante que para la clase **Giro** es infinito
Procedure **Cambio_Origen**(O: **In Arco**);
// Cambia El Arco de origen del giro por O
Procedure **Cambio _Destino** (D: **In Arco**);
// Cambia El Arco de destino del giro por D
Procedure **Cambio _Costo** (C:**In Real**);
// Cambia el costo del giro por C
Procedure **Eliminar** ();
//Elimina un giro

Lista_De_Giros

Class **Lista_De_Giros** is **List(Giro)**

Arco

Class **Arco** Inherits **Objeto_De_Mapa**

uses **Esquina**
uses **Real**
uses **Entero**
uses **Boolean**
uses **Layer**
uses **Lista_de_Impedancias**
uses **Lista_de_Banderas**
uses **Lista_de_Puntos**

Exports

Procedure **Crear**(L: **In Layer**, Enc, Exist: **In Boolean,** Ori, Des: **In Esquina,** Id_Ext:**In Entero,** F:**In Entero,** Imp:**In Lista_de_Impedancias**, Band: **In Lista_de_Banderas**, Puntos: **In Lista_de_Puntos**);
// Crea un Arco en que la esquina de origen es Ori, La esquina de destino es Dest, Cuya flecha va en el sentido de la digitalización, en el contrario o en ambos según F valga 1, -1 o 0 respectivamente, Con lista de impedancias Imp, Lista de banderas Band y Lista de puntos asociados Puntos, Que pertenece al Layer L, que esta encendido sii Enc es True y que existe lógicamente sii Exist es True

Function **Su_Origen**():**Esquina;**
// Devuelve La esquina de origen del Arco
Function **Su_Destino** ():**Esquina;**
// Devuelve La esquina de destino del Arco
Function **Su_Lista_de_Impedancias**(): **Lista_De_Impedancias**
// Devuelve la Lista de impedancias del arco, esto es una lista de los diversos costos que pueden usarse luego para los algoritmos. Algunos de estos costos están predeterminados y son:

- Largo, que se calcula como la distancia euclídea entre la Esquina de origen y la esquina de destino
- Tiempo, que se recibirá del usuario de la estructura.

Function **Su_Lista_de_Banderas**(): **Lista_De_Boolean**
// Devuelve la lista de banderas del arco, esto es una lista de las diversas banderas que pueden usarse luego en los algoritmos para tomar en cuenta o descartar un arco
Function **Su_Lista_de_Puntos_Asociados**(): **Lista_De_Puntos**
// Devuelve la lista de Puntos Asociados del arco, esto es una lista de todos los puntos (depósitos, clientes, etc) que están sobre el arco (por ser su distancia al arco menor que un épsilon dado, y menor que la distancia a cualquier otro arco)
Procedure **Cambio_Origen**(O: **In Esquina**);
// Cambia La esquina de origen del Arco por O

Procedure **Cambio_Destino** (D: **In Esquina**);
// Cambia La esquina de destino del Arco por D
Procedure **Cambio_Lista_de_Impedancias**(I:**In Lista_De_Reales**));
// Cambia la lista de impedancias del arco por I
Procedure **Cambio_Lista_de_Banderas**(B:**In Lista_De_Boolean**));
// Cambia la lista de Banderas del arco por B
Procedure **Cambio_Lista_de_Puntos_Asociados**(P:**In Lista_De_Reales**));
// Cambia la lista de Puntos asociados del arco por P
Procedure **Eliminar** ();
//Elimina un Arco

Lista_De_Arcos

Class **Lista_De_Arcos** is **List(Arco)**

Punto

class **Punto**
// Corresponden con los clientes y depósitos de la estructura

uses **Punto_Cartesiano**
uses **List**
uses **Entero**
uses **string**
uses **Lista_De_Demandas**
uses **Lista_De_Boolean**
uses **Lista_De_Arcos**
uses **Lista_De_Giros**
uses **Info_Punto**

Exports

Procedure **Crear**(Ub: **in Punto_Cartesiano,** Inc : **In Lista_De_Arcos,** Inf: **In Info_Punto,**Gir: **In Lista_De_Giros**);
// Crea una punto con ubicación (posición geográfica) ub, Lista de arcos incidentes Inc, Información del punto Inf y Lista de giros gir. La lista de Giros es la lista de costos de girar de un arco (incidente hacia adentro) a el punto hacia otro (incidente hacia afuera de la misma punto. Si un par de arcos no aparece en dicha lista se asume costo cero. Si es un giro imposible se asignará un costo infinito(la constante de la clase giro).
Function **Su_Ubicacion**():**Punto_Cartesiano;**
// Devuelve La ubicación del punto
Function **Su_Id_Externo** ():**Entero;**
// Devuelve el identificador externo del punto
Function **Su_Id** ()**:Entero**
// Devuelve el identificador interno del punto
Function **Su_Informacion** ()**Info_Punto;**
// Cambia La información del punto por Inf
Function **Sus_Demandas** ():**Lista_De_Demandas;**
// Si es un cliente Devuelve La Lista de demandas del punto, si es un depósito, podrá representar la lista de Productos disponibles (o algo así).

Function **Sus_Banderas** ():**Lista_De_Boolean;**
// Devuelve La Lista de banderas del punto
Function **Su_Nombre** (): **String**;
// Devuelve el nombre del cliente o depósito
Function **Su_TIPO**()**boolean;**
// Devuelve true si es cliente y false si es depósito
Procedure **Cambio_Ubicacion**(U: **In Punto_Cartesiano**);
// Cambia la ubicación del punto por U
Procedure **Cambio_Id_Externo** (Id:**In Entero);**
// Cambia el identificador externo del punto por Id
Procedure **Cambio_Informacion** (Inf: **In Info_Punto**);
// Cambia La información del punto por Inf
Procedure **Cambio_Demandas** (D:**In Lista_De_Demandas**);
// Cambia La Lista de demandas del punto por D
Procedure **Cambio_Banderas** (B: **In Lista_De_Boolean)**);
// Cambia La Lista de banderas del punto por B
Procedure **Cambio_Nombre** (N: **In string**);
// Cambia el nombre del cliente o depósito por N
Procedure **Cambio_TIPO**(T: **In boolean)**
// Marca que es cliente si T es true y depósito si es false
Procedure **Eliminar** ();
//Elimina una punto

Lista_De_Puntos

Class **Lista_De_Puntos** is **List(Punto)**

Ruta

Class **Ruta**

uses **Lista_De_Objetos_De_Mapa**
uses **Lista_De_Boolean**
uses **Lista_De_Reales**

exports

Procedure **Crear**(Loc: **In Lista_De_Objetos_de_Mapa**, Cos: **In Lista_De_Reales**,
F: **In Lista_De_Boolean**);
// Crea una ruta Con lista de locales y depósitos Loc, Lista de costos Cos y Lista de Flags F

function **Su_Lista_De_Objetos_de_Mapa**():**Lista_De_Objetos_de_Mapa**
// Devuelve la lista de locales por los que pasa la ruta.

function **Su_Lista_De_Costos**():**Lista_De_Reales**
// Devuelve la lista de costos asociados a la ruta.

Procedure **Cambio_Lista_De_Objetos_de_Mapa** (L: **In Lista_De_Objetos_de_Mapa**);

// Cambia la lista de locales por los que pasa la ruta por L

Procedure **Cambio_Lista_De_Costos**(L: **In Lista_De_Reales**);
// Cambia la lista de costos asociados a la ruta por L

Procedure **Cambio_Lista_De_Flags**(L: **In Lista_De_Boolean**);
// Cambia la lista de flags asociadas a la ruta por L

Procedure **Eliminar**();
/* Elimina una ruta */

Vehículo

class **Vehiculo**

uses **Lista_De_Costos**
uses **Lista_De_Flags**
uses **Entero**
uses **String**
uses **Real**

exports

Procedure **Crear** (L_Flag: **In Lista_De_Flags**, L_Cap: **In Lista_De_Costos**, T_Aten, TWMin, TWMax: **In Real**, IdExt: **In Entero**, Nom: **In String**);
// Crea un vehículo con. Lista de Flags L_Flag, lista de capacidades L_Cap, tiempo de //atención T_Aten, comienzo de la ventana de tiempo TWMin, fin de la ventana de //tiempo TWMax, Id_Esterno IdExt y nombre Nom

Function **Su_Lista_De_Capacidades**(): **Lista_De_Costos**;
// Devuelve la lista de capacidades.

Function **Su_Lista_De_Flags**():**Lista_De_Flags**;
// Devuelve la lista de flags.

Function **Su_Tiempo_de_Atencion**(): **Real**;
// Devuelve el tiempo de atención.

Function **Su_Twmin**():**Real**;
// Devuelve el comienzo de la ventana de tiempo.

Function **Su_TWmax**():**Real**;
// Devuelve el fin de la ventana de tiempo.

Function **Su_Id_Interno**():**Entero**;
// Devuelve el identificador interno.

Function **Su_Id_Externo**():**Entero**;
// Devuelve el identificador externo.

Function **Su_Nombre**(): **String**;
// Devuelve el nombre del Vehículo.

Procedure **Cambio_Nombre**(Nom: **In String**);
// Asigna al Vehículo el nombre Nom.

Procedure **Cambio_Lista_De_Capacidades**(L: **In Lista_De_Costos)**;
// Cambia la lista de capacidades por L.

Procedure **Cambio_Lista_De_Flags**(L: **In Lista_De_Flags)**;
// Cambia la lista de flags por L.

Procedure **Cambio_Tiempo_de_Atencion**(T: **In Real)**;
// Cambia el tiempo de atención por T.

Procedure **Cambio_Twmin**(T: **In Real)**;
// Cambia el comienzo de la ventana de tiempo por T

Procedure **Cambio_TWmax**(T: **In Real)**;
// Cambia el fin de la ventana de tiempo por T.

Procedure **Cambio_Id_Externo**(Id: **In Entero)**;
// Cambia el identificador externo por Id.

Procedure **Eliminar**();
// Elimina un vehículo.

Estructura

Class **Estructura**

uses **Layer**
uses **Boolean**

Procedure **Crear**(L: **In Lista_De_Layers**,E: **In Lista_De_Esquinas,**V: **In Lista_De_Vehiculos**);
//Crea una Estructura Con Lista de Layers L, Lista de Esquinas E y Lista de vehículos V. Con la lista de Layers y/o con la lista de esquinas se puede tener acceso a todo el mapa.
Function **Su_Lista_De_Layers**: **Lista_De_Layers;**
// Devuelve La lista de Layers
Function **Su_Lista_De_Esquinas**: **Lista_De_Esquinas;**
// Devuelve La lista de esquinas
Function **Su_Lista_De_Vehiculos**: **Lista_De_Vehiculos;**
// Devuelve La lista de Vehiculos
Procedure **Cambio_Lista_De_Layers** (L: **In Layer**);
// Cambia la lista de Layers por L
Procedure **Cambio_Lista_De_Esquinas** (E: **In Layer**);
// Cambia la lista de Esquinas por E

Procedure **Cambio_Lista_Vehiculos** (V: **In Layer**)**;**
// Cambia la lista de vehiculos por V
Procedure **Eliminar** ()**;**
//Elimina una Estructura

Testeo

PLAN DE TESTEO

Introducción

En este capítulo se presenta el plan de testeo para la estructura de datos que servirá como biblioteca para correr algoritmos de ruteo y asignación sobre ella, así como de la Interface con ArcView. Este documento se divide en dos partes. La primera es una breve reseña de cómo se haría un testeo de software en general. La segunda es el tipo de testeo realizado en el problema concreto.

Testeo de Software

Presentamos aquí un plan genérico de verificación que permite realizar un testeo de un sistema de la complejidad del nuestro, si bien es probable que exceda el tiempo y los recursos de que se dispone en el entorno del Taller V. Se enfoca el plan teniendo en cuenta las facilidades que ofrece implementar la estructura en un lenguaje orientado a objetos.
Se realiza un testeo de Caja Negra a cada módulo y luego un testeo de integración. Estas pruebas se realizan con juegos de datos pequeños de modo de detectar fácilmente la mayoría de los errores. Por último se realiza un testeo de sobrecarga, que en caso de permitirlo el tiempo, sigue los mismos pasos de lo anteriormente detallado, y en caso contrario, solamente se realiza el testeo global del módulo.

Testeo de la estructura

Tomando en cuenta las dificultades de interface que presenta nuestro sistema se procede de la siguiente manera:
Primero se testea la interface, una vez aceptada la misma como válida, se utiliza la misma para generar los datos de prueba para el resto de los módulos. Para ello se utiliza ArcView y la Interface implementada para generar los archivos de texto que levanta nuestra estructura. Entendemos que sería preferible generar los datos independientemente, pero este tipo de testeo llevaría mucho más tiempo y/o recursos humanos que los que disponemos. Para compensar este déficit, se revisa con cuidado los archivos de texto resultantes.
Para el testeo modular, sería bueno, teniendo en cuenta las características de la estructura, realizar un testeo incremental, esto es: Se testean primero los módulos básicos, y aquellos para los que se pueda crear fácilmente interfaces de prueba. Luego de validados los mismos, se van utilizando para probar uno a uno los demás módulos. Este tipo de testeo se justifica por la naturaleza del diseño, por la cual, los módulos más avanzados utilizan fuertemente los más básicos, de modo que la opción alternativa implicaría simular cada uno de los módulos utilizados por el módulo más avanzado.
En este caso particular, tanto cumpliendo con los requerimientos de la Empresa, como tomando en cuenta el tiempo y recursos humanos disponibles, sólo se realiza un testeo global de la estructura mediante la implementación de tres algoritmos sobre la misma y verificando que los resultados de los mismos sean correctos. Sólo se prueba

individualmente algún módulo en caso que se detecten errores durante el testeo, como forma de identificar el módulo que los está causando.

Testeo de interfaces

Se utiliza para esto partes del mapa de Atlanta (o todo el mismo), utilizando el mapa y las funcionalidades de ArcView para seleccionar una parte del mismo.

Testeo de integración

La estructura funciona como biblioteca, de tal modo que se pueda programar algoritmos de Ruteo utilizando la misma.
Probar la integración es equivalente a probar que la estructura funciona como tal y que es posible implementar algoritmos de ruteo sobre la misma, controlando que su implementación pueda hacerse en un tiempo aceptable y que los algoritmos resultantes funcionen también en tiempos aceptables.

Algoritmos utilizados para probar la estructura

- Dijkstra
- Asignación
- TSP
- TSP sobre asignación

Para ver los detalles de los tres últimos ver Análisis de Requerimientos.

Plan de Regresión

En caso de detectarse un error en la estructura, se busca el módulo que lo causa, se arregla y luego se retestea la estructura, concentrándose especialmente en las partes que utilizan el módulo arreglado.

RESULTADOS DEL TESTEO

Como resultado del testeo, se pueden sacar las siguientes conclusiones:
- Se cumplió con los objetivos, dado que la estructura funciona y permite la implementación de Algoritmos sobre ella.
- Tanto en el caso de la implementación de Algoritmos de Ruteo, como en el tiempo de corrida de los mismos se obtienen resultados en tiempos muy aceptables.

TIEMPOS DE CORRIDAS

Estos tiempos fueron tomados trabajando con el Mapa de Atlanta que viene con Arcview y con información adicional proporcionada por la Empresa.

Características de los Datos de Prueba

Se utilizó el mapa completo de la Ciudad de Atlanta (que viene con Arcview).

La máquina utilizada para las pruebas tiene las siguientes características:
- Procesador: Pentium 120
- RAM: 40MB
- S.O.: Windows 98 (recordar que al ser multitarea no se puede estar seguro de contar con el 100% del procesador)
- Se corrió en entorno de diseño (que en general es más lento que el ejecutable) del Lenguaje de Programación MS Visual C++ Versión 6.0.

Entre otras cosas se verificó que el mapa de Atlanta con que se contaba tenía imprecisiones de digitalización en el sentido de que algunas calles que en la realidad se cortan, para una precisión muy pequeña, no era así. Por esto se realizaron pruebas para dos posibles precisiones (recordar que se definió una precisión para la cual dos puntos que están a menos de esa precisión se consideran iguales). La precisiones probadas fueron.:
- 0, en cuyo caso el mapa queda desconexo, haciendo que algunos algoritmos deban recorrer toda la estructura para verificar que no se tiene acceso.
- 0.0005, con esta precisión la mayor parte de las desconexiones anteriores desaparecieron.

Los tiempos fueron calculados utilizando las funciones time y difftime de c++ (cuya mayor precisión son segundos), evitando el uso de otras funciones disponibles en Visual C++, por no ser compatibles con los compiladores para Unix, interfiriendo con que la estructura sea multiplataforma.

Tiempos obtenidos

Parte	**Precisión 0**	**Precisión 0.0005**
➢ Interface (2107 arcos)		
➢ Calles	9 seg.	9 seg.
➢ Depósitos (2 locales)	0 seg.	0 seg.
➢ Clientes 1 (9 locales)	0 seg.	0 seg.
➢ Clientes 2 (8 locales)	0 seg.	0 seg.
➢ Carga de la estructura (2107 arcos)		
➢ Calles	39 seg.	32 seg.
➢ Depósitos (2 locales)	0 seg.	0 seg.
➢ Clientes 1 (9 locales)	1 seg.	1 seg.
➢ Clientes 2 (8 locales)	1 seg.	1 seg.
➢ Implementación de Algoritmos Obs: en el caso de TSP y asignación los algoritmos eran desconocidos, mientras que Dijkstra ya era conocido		
➢ Dijkstra	5 horas	
➢ Asignación	8 horas	
➢ TSP	6 horas	
➢ Corrida de Algoritmos		
➢ Dijkstra (19 loc.–2107 aristas)	3 seg.	1 seg.
➢ Asignación (2 dep., 17 cli.)	96 seg.	63 seg.
➢ TSP (19 locales)	752 seg.	514 seg.

Conclusiones y trabajo futuro

CONCLUSIONES

Se cumplió con los objetivos planteados, implementando una Estructura de Datos Genérica que soporta cualquier algoritmo de solución del Problema de Ruteo de Vehículos. Se puede apreciar a través de los resultados del Testeo que tanto levantar la estructura como los algoritmos que corren sobre ella funcionan en **tiempos muy aceptables**.
Se confirmó que C++ es un lenguaje adecuado para cumplir con los objetivos planteados, tanto por su eficiencia y generalidad, como por su posibilidad de acceso a recursos de bajo nivel y de levantar y recorrer archivos binarios como tales.
Se observa que la única forma de utilizar todas las potencialidades de la estructura es conocer el lenguaje de programación C++. En caso de desearse que la estructura sea utilizada por un usuario que desconozca C++ habría que programar en C++ una interface con este usuario, perdiendo gran parte de la generalidad de la estructura.

PRINCIPALES DIFICULTADES ENCONTRADAS

- **Interface**

  Cabe destacar la dificultad presentada por la implementación de la Interface (que debía ser un producto auxiliar al proyecto pese a su interés en si misma). Las dificultades de implementación de la Interface se debieron tanto a la dificultad de manejo de los formatos involucrados, como a la falta de documentación sobre los mismos, que requirió una búsqueda mucho más exhaustiva de lo que se supuso originalmente.

- **Estructura**
  - Análisis y diseño:
    - Tomó tiempo el análisis de posibilidades para la elección de la estructura, que permitiera cumplir con los requisitos y la elección de una en concreto.
    - También implicó un estudio delicado el hecho de tener que asociar los puntos a alguna calle, dado que al "perder" la geografía, saber si un punto está lo bastante cerca de una arista o determinar las aristas que distan de un punto menos que una distancia dada implica recorrer toda la estructura.
    - Las reglas de tránsito, semáforos, etc. requirieron decisiones importantes sobre qué asociar a qué.
  - Implementación: La generalidad de la Estructura está muy ligada, en C++ al uso de la herencia y el polimorfismo, que en este lenguaje implican mucho uso de punteros con las dificultades de implementación que eso conlleva.

- **Comunicación**
  - Al tratarse de un Proyecto de Fin de Carrera, como es de suponer, los integrantes del Grupo trabajamos en diversos horarios, lo que dificulta el trabajo juntos así

como la interacción con el representante de la empresa, ya que tanto en horarios como en ubicación, coincidir fue difícil.

- La falta de conocimiento (más allá del básico) del Grupo sobre Ruteo de Vehículos potenció la necesidad de mantener un fluido contacto con el usuario, tanto para definir detalladamente los requerimientos, como para tomar las decisiones (tanto de diseño como de implementación), que debido a nuestra falta de experiencia en el tema específico, así como por no contar con la ayuda "on-line" del usuario se presentaron en tiempo de implementación. Esto produjo algunas demoras tanto en el Análisis y Diseño como en la Implementación.

TRABAJO FUTURO

- Como se menciona en el análisis de requerimientos, la fórmula utilizada como distancia puede plantear algunos problemas e inexactitudes, si bien en general funciona bastante bien. Para que no de lugar a errores.
  Lo que debería calcularse para asignar correctamente los puntos a las aristas es la arista a que tiene como distancia euclídea al punto la solución de lo siguiente:

$$\begin{cases} \min d(x,a) \\ \text{s.a. } d(x,a) < D \text{ y la } \prod_a (x) \text{ cae sobre a.} \end{cases}$$

  siendo $d(x,a)$ la distancia (euclídea) del punto x a $\prod_a (x)$ (su proyección ortogonal sobre la recta determinada por a) y D un parámetro dado que determina si un punto puede o no estar sobre una arista (ver Requerimientos de la empresa)

  Esta mejora no fue implementada por falta de tiempo.

- Quedaría por implementar una salida de datos (rutas) y una interface que permita levantar la información con ARCVIEW de modo de dar una respuesta gráfica al problema.

- También podría continuarse el trabajo por la línea de las Interfaces, dado que no hay disponible una interface con los formatos de ARCVIEW, podría implementarse la interface inversa, es decir, que parta del archivo de texto con el formato dado y devuelva los archivos .shp, .shx y .dbf que levanta ARCVIEW

Bibliografía

LIBROS Y PAPERS

[1] Lawrence Bodin, Bruce Golden, Arjang Assad and Michael Ball: *Routing and scheduling of vehicles and crews. The state of the art.* Pergamon Press, Oxford – New York - Toronto – Sydney – París – Frankfurt, 1983

[2] Bruce Golden and Arjang Assad: *Vehicle Routing: Methods and studies.* Elsevier Science Publishers B.V. (North Holland), 1988

[3] Especificación de Requerimientos, provisto por la empresa ICA

[4] ESRI Shapefile Technical Description
An ESRI White Paper-July 1998

[5] Javier Ceballos: *Curso de programación en C++*. Addison-Wesley, 1991
Visual C++ 6.0: *Programmers Guide*. Microsoft Press.

[6] Bjarne Stroustrup: *Introduction to C++ programming language.*

[7] Apuntes de Investigación Operativa
Oficina de Publicaciones CEI

[8] Patrick Valduriez, Project Rodin (INRIA): *Some hints to improve Writting of technical papers*.

[9] Carlo Ghezzi, Mehdi Jazayeri, Dino Mandrioli.: *Fundamentals of Software Engeneering.*. Prentice-Hall,

[10] Diseño de la estructura (Ignacio Alonzo y Fernanda Burgueño)

[11] Análisis de Requerimientos (Ignacio Alonzo y Fernanda Burgueño)

PÁGINAS WEB Y LIBROS EN PANTALLA

[12] Libros en Pantalla de Microsoft Project

[13] http://www.evm.khstu.ru/Students/FAQ/Delphi/uddf/pages/dbase.htm

[14] http://www.fing.edu.uy/~taller3/principal.html

[15] www.usgs.gov/research/gis

[16] www.census.gov

[17] www.sun.acsu.buffalo.edu (David Mark Homepage)

[18] www.magnus.acs.ohio-state.edu (Duane F. Marble Homepage)

[19] http://hissa.ncsl.nist.gov/~black/CRCDict/HTML/nphard.html

[20] http://hissa.ncsl.nist.gov/~black/CRCDict/HTML/np.html

[21] http://hissa.ncsl.nist.gov/~black/CRCDict/HTML/turingmachin.html

[22] http://hissa.ncsl.nist.gov/~black/CRCDict/HTML/npcomplete.html

[23] http://hissa.ncsl.nist.gov/~black/CRCDict/HTML/polynomialtm.html

[24] http://www.routesmart.com/

[25] http://www.semcor.com/gis/solutions/type/route/route.html

[26] http://www.semcor.com/gis/solutions/type/route/tracking.html

[27] http://www.esri.com/library/whitepapers/addl_lit.html

[28] http://www.esri.com/library/whitepapers/pdfs/data_pub.pdf

[29] http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

Apéndice 1:

REQUERIMIENTOS DE LA ESTRUCTURA DE DATOS PARA PROBLEMAS GENÉRICOS DE RUTEO.

Problemática

Los problemas de ruteo tienen dos componentes importantes, la estructura de datos sobre la cual se corren los algoritmos, y los algoritmos en sí. En nuestro caso la preocupación es la estructura de datos para los algoritmos. Cada algoritmo en particular tiene asociada una estructura que se adecua al mismo de la mejor manera, pero cualquier cambio en dicho algoritmo hace que la estructura quede obsoleta o que halla que volver a diseñarla, por lo que es importante una buena estructura de datos que soporte una gran grama de problemas.

Objetivo final

Como objetivo del Taller V se busca tener una estructura de datos con sus correspondientes funciones de manejo, que permite montar sobre la misma todos los componentes de un problema de "Ruteo de Vehículos", resolverlo y recuperar los resultados.

Este documente de requerimiento está organizado de la siguiente manera: para todos los requerimientos existe una parte común a los grupos de C++ y en ArcView, seguida (para los casos en que exista alguna diferencia) de los requerimientos diferenciados por estructura de datos.

Entrada de Datos

Los datos los vamos a dividir en dos categorías, los datos geográficos y no geográficos.

Geográficos

Son aquellos que tienen una componente geográfica. Están dados en formato shapefile de: líneas y puntos (el shapefile de polígonos no es un dato de entrada).

Shapefile de Líneas:

Los shapefile de líneas corresponden a la red sobre la cual se realizan los cálculos de ruteo. Debemos poder cargar varios shapefile de líneas uno sobre los otros, por ejemplo: un shapefile de calles, otro de avenidas, otro de puentes, etc. y que la unión de éstos forme la red sobre la cual vamos a colocar los puntos de interés (depósitos, clientes, etc.).

En el archivo **.dbf* de cada shapefile, existe un conjunto de datos que deben ser tenidos en cuenta por la estructura de datos, estos son:

- *Id:* Identificador del usuario, es un entero.

- *Id_interno:* este identificador es usado y manejado internamente por los algoritmos.
- *Largo:* largo del arco, es un real.
- *Tiempo:* tiempo de viaje del arco, es un real.
- *Flecha:* indica el flechamiento de las calles, este campo puede tener tres valores:
  0: cuando la calle no esta flechada
  1: cuando el flechamiento es en el sentido de la digitalización
  -1: cuando el flechamiento es en sentido contrario a la digitalización.
- *Habilitado:* indica si se encuentra habilitado dicho arco:
  0: no esta habilitado
  1: está habilitado.

Estos campos no tienen porque encontrarse en los datos, ya que por ejemplo si la red no es flechada, el campo *flecha* es superfluo. Lo mismo sucede con el *Id*, esta es una identificador del usuario por lo que puede o no encontrarse, además de encontrarse no se asegura que los datos sean confiables ya que el manejo de los mismos lo hace el usuario y no la estructura.
El campo *Habilitado* corresponde al conjunto de las "flags" que puede tener un arco.

Datos adicionales al shapefile de líneas

Hay otras consideraciones que vamos a realizar, en este caso especificado anteriormente existen un par de campos que indican el "peso" del arco, éstos son *Largo* y *Tiempo*, de todas formas puede haber un conjunto mucho mayor de "pesos" en cada arco, por ejemplo: varios tiempos indicando velocidades distintas de viaje según horas del día o días de la semana. Para estandarizar los nombres de los campos y sus datos, junto con el shapefile de líneas, tienen como entrada un archivo ASCII que indica que campo tiene que información, este archivo se llama como el shapefile pero con extensión *inf*.

Este archivo tiene el siguiente formato:

[IdUsuario]
Nombre del campo de ID usario: este puede ser vacío
[Flecha]
Nombre del campo de flechamiento: puede ser vacío.
[Habilitado]
Nombre del campo de habilitacion: puede ser vacío.
[Largo]
Nombre del campo largo: puede ser vacío.
[Tiempo]
Nombre del campo de tiempo: pude ser vacio
[peso1]
Nombre del campo de peso1
[peso2]
Nombre del campo de peso2
...
...
[pesoN]
.....
[Flag1]
Nombre del campo de la flag1

[Flag2]
Nombre del campo de la flag1
.....
[FlagM]

Al menos uno de los campos que corresponde al "peso", o sea: *largo, tiempo, peso1..peson,* tiene datos, porque de lo contrario los algoritmos de Ruteo no tiene un consto de transporte.

Los tipos de datos que debe soportar cada campo descripto anteriormente es el siguiente:

[IdUsuario]	entero
[Flecha]	entero
[Habilitado]	entero(igual que una flag)
[Largo]	float/double
[Tiempo]	float/double
[peso1..n]	float/double
[Flag1..n]	entero

Shapefile de puntos

Los shapefile de puntos corresponden a los puntos de interés, como por ejemplo depósitos, clientes, paradas obligatorias, etc.

De la misma forma que los shapefile de líneas tienen datos de importancia en el archivo *dbf*, los shapefile de puntos tiene un conjunto de datos de relevancia.

- *Id:* Identificador del usuario, es un entero.

De la misma forma que en el shapefile de líneas, estos datos pueden o no encontrarse.

Datos adicionales al shapefile de puntos

Para estandarizar los nombres de los campos y sus datos, junto con el shapefile de puntos, tienen como entrada un archivo ASCII que indica que campo tiene que información, este archivo se llama como el shapefile pero con extensión *inf.*

Este archivo tiene el siguiente formato:

[IdUsuario]
Nombre del campo de ID usario: este puede ser vacío
[Demanda/Capacidad inicial 1]
Nombre del campo de demanda o capacidad: puede ser vacío.
[Demanda/Capacidad inicial 2]
.....

[Demanda/Capacidad inicial K]
.....
[Flag1]
Nombre del campo de la flag1
[Flag2]
Nombre del campo de la flag1
.....
[FlagM]
.....
[Nombre]
Nombre del campo "Nombre" del punto

Los tipos de datos que debe soportar cada campo descripto anteriormente es el siguiente:

[IdUsuario]	entero
[Demanda/Capacidad inicial 1..k]	float
[Flag1..m]	entero
[Nombre]	string

No Geográficos

Los datos que no son geográficos son aquello que no tienen una componente geográfica tales como las reglas de tránsito, los vehículos, etc.

Todos estos datos están dados en archivos ASCII, con cabezal y delimitados por comas de nombres:

- *Reglas.txt*: reglas de tránsito
- *Vehículos.txt:* los vehículos
- *Semáforos.txt:* semáforos

Reglas de tránsito

Indicar las reglas de tránsito, o sea, prohibiciones de doblar, costo de hacer ciertas maniobras en una esquina, realizar un giro, etc. Estos datos están en un archivo ASCII de nombre *Reglas.txt* y en el siguiente formato:

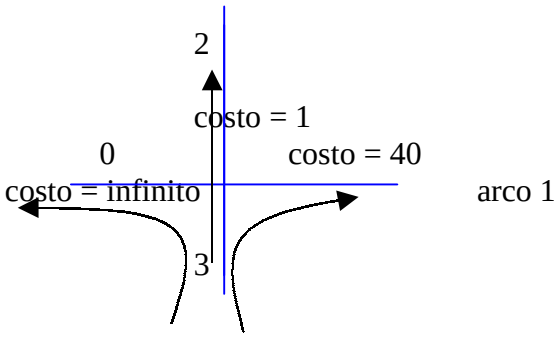
"Id_arco_origen","Id_arco_destino","costo"	cabezal
Id_arco_origen, Id_arco_destino, costo	datos de cada linea
.....	
.....	

el cabezal del archivo puede tener otros nombres, pero el orden en el cual se encuentran los campos no cambia.

Los *id* de los arcos, son los usados por el usuario.

No tiene por que haber una regla de tránsito para cada esquina, todo lo contrario, lo normal es que algunos cruces de calles tengan reglas de tránsito.

Ejemplo del archivo:



supongamos que tenemos esa boca calle, con los arcos, 0..3, y las prohibiciones que se indican con sus costos, el archivo tendrá la siguiente información:

Id_arco_origen,	*Id_arco_destino,*	*costo*
3,	2,	1
3,	0,	infinito (lo representamos con un numero grande)
3,	1,	40

Por defecto el costo de un maniobra es 0.

Semáforos

Los semáforos tienen una componente geográfico, su ubicación, pero en nuestros problemas no nos interesa donde se encuentras posicionados sino las demora que genera cruzar uno. Estos datos están en un archivo ASCII de nombre *Semaforos.txt* y tiene el mismos formato que el de *Reglas*, con la diferencia que el costo es el tiempo (o su equivalente en distancia) que se pierde por cruzar dicho semáforo en una dirección. Este "peso" que tienen los semáforos debe ser tomado en cuenta por las funciones que implementan sobre la estructura (más adelante profundizaremos en ello).

Vehículos

Pueden o no tener una componente geográfico, ya que dentro de los datos de los vehículos pueden tener la coordenada de donde se encuentran al inicio del problema.
Los datos que pueden tener los vehículos son los siguientes:

- *Id*: identificador del usuario (entero)
- *Capacidad:* capacidad del vehículo (real), los vehículos pueden tener mas de un compartimento.
- *Twmin:* horario a partir del cual puede ser usado (entero - minutos)
- *Twmax:* horario a partir del cual es vehículo queda fuera se servicio. (entero – minutos)
- *Nombre:* nombre del vehículo (string)

Como cada uno de estos campos puede o no encontrarse, además del archivo de vehículos, hay un archivo *vehiculos.inf* que contiene la descripción de los datos contenidos en *vehiculos.txt*.

[IdUsuario]
Nombre del campo de ID usario: este puede ser vacío
[Capacidad 1]
Nombre del campo de capacidad del vehículo: puede ser vacía, indicando capacidad infinita.
[Capacidad 2]
.....
[Capacidad K]
.....
[TWmin]
Nombre del campo que indica el comienzo de la Ventana de Tiempo del vehículo
[TWmax]
Nombre del campo que indica el fin de la Ventana de Tiempo del vehículo
[TiempoAtención]
Nombre del campo con el tiempo de atención
[Flag1]
Nombre del campo de la flag1
[Flag2]
Nombre del campo de la flag1
.....
[FlagM]
.....
[Nombre]
Nombre del campo con el "nombre" del vehículo

Los tipos de datos que debe soportar cada campo descripto anteriormente es el siguiente:

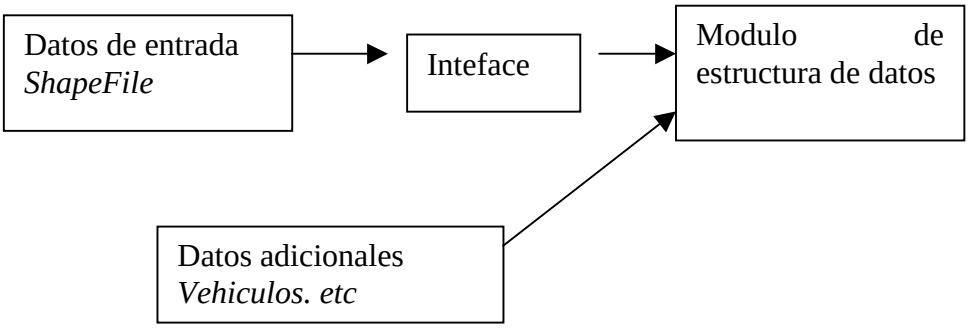
[IdUsuario]	entero
[Capacidad1..k]	float
[TWmin]	float
[TWmax]	float
[TiempoAtencion]	float
[Flag1..m]	entero
[Nombre]	string

Observaciones

Los nombres de los archivos de datos adicionales (vehículos, semáforos, etc.) no tienen por que ser los descriptos anteriormente, deben implementar las funciones necesarias para cargar esta información, teniendo como parámetro el nombre del archivo.

Estructura de datos en C++

Los que deben realizar la estructura de datos en C++, tienen una pequeña diferencia en la entrada de datos. Deben realizar una interface bien definida entre los shapefile y el módulo de la estructura de datos, de forma de independizarse del ArcView, y del formato shapefile.



Esta interface toma los shapefile y los transforma en un archivo ASCII delimitado por comas, el cual es la entrada a la estructura de datos.

Salida de Datos

La salida de datos se realiza en archivos de formato ASCII con cabezal delimitado por comas.
Los datos más importantes en una estructura de datos de este tipo es el estado de los objetos y en que orden deben ser recorridos, no debemos dejar de tener presente que la estructura de datos está implementada para soportar problemas de ruteo, retornando en forma clara la solución.

Objetos en la estructura

Listados de todos aquellos objetos con sus propiedades que se encuentran en la estructura, se debe poder filtrar los objetos según las flags.

ArcView

ShapeFile con los datos, por ejemplo una ruta debe ser un shapefile de polilineas con un registro por cada tramo del recorrido (nos referimos a tramo a aquel que va de un cliente a otro), y la información asociada a él.

C++

Salida de datos en un formato estándar (ASCII)

Funcionalidades

Sobre los ShapeFile de líneas (RED)

Se deberá poder cargar varios shapefile de líneas (redes) uno sobre los otros, por ejemplo: un shapefile de calles, otro de avenidas, otro de puentes, etc. y que la unión de éstos forme la red sobre la cual vamos a colocar los puntos de interés (depósitos, clientes, etc.).

Atributos adicionales de una RED

- *Capa:* entero que indica el número de capa en la cual se cargó la red, el usuario indica que número de capa le quiere dar.

  Del conjunto de redes cargados en distintas capas, se puede habilitar o deshabilitar un conjunto de las mismas para trabajar en ese momento. Se puede trabajar con la red cargada en cada capa por separado o en el conjunto que el usuario desea.

Deben permitir agregar atributos adicionales de forma fácil a la estructura de datos, dejando en claro las operaciones que se deben realizar para operar con los mismos.

Seteo de atributos de la RED

- *Id*: permite modificar el id del usuario
- *Flechamiento*: permite cambiar, habilitar y deshabilitar el flechamiento de un arco, también habilitar y deshabilitar todo el flechamiento de la RED.
- *Flags:* todas las flags, incluso "habilitado" pueden ser seteadeas.
- *Pesos:* todos los "pesos" incluso "Largo y Tiempo" pueden ser seteados

Consultas en la RED

- *Id* de un arco
- *Flechamiento* de un arco
- Si se encuentra habilitado o no el flechamiento de la RED
- *Flag***n** de un arco
- *Peso***n** de un arco

Sobre los ShapeFile de puntos

Se deberá poder cargar varios shapefile de puntos (depósitos, clientes, etc.) todos sobre las redes previamente cargadas. Se puede indicar (a través de la capa) en que red se va a cargar, o en que redes se van a cargar. Distinto shapefile de puntos pueden cargarse en redes de diferentes capas.

Los puntos se pueden encontrar en cualquier punto sobre la red, e incluso fuera de ella. En la mayoría de los casos los puntos se van a encontrar fuera de la red, debido a los errores de datos, de todas formas debemos de alguna forma aproximar los puntos que se encuentran fuera de la red, a la red. Debo poder (como usuario), dar un rango (por ejemplo distancia), a partir del cual los puntos que se encuentren a una distancia mayor que ésta dada, queden fuera de la red, o sea, no son alcanzables.

Atributos adicionales de un punto

- *Capa:* entero que indica el número de capa en la cual se cargó, el usuario indica que número de capa le quiere dar.
- Algunos objetos como los clientes, pueden tener atributos adicionales tales como: *horaDeLlegada, horaDeAtención, horaDeSalida, CapaicadaODemandaRestantelN*
- *Visitado:* este pertenece a un conjunto de flags internas
- *Ruteado:* ídem a lo anterior.
- *Asignado:* ídem a lo anterior.

Deben permitir agregar atributos adicionales de forma fácil a la estructura de datos, dejando en claro las operaciones que se deben realizar para operar con los mismos.

Seteo de atributos de los Puntos

- *Id*: permite modificar el id del usuario
- *Flags:* todas las flags, incluso "habilitado" pueden ser seteadeas.
- *Pesos:* todos los "pesos" incluso "Largo y Tiempo" pueden ser seteados
- *Capacidad***n:** cambiar las capaciadades
- *HoraDeLlegada*
- *HoraDeAtención*
- *HoraDeSalida*
- *Capacidad_Demanda_Restante***n:** cambiar la demanda o la capacidad N restante.

Consultas en los Puntos

- *Id* de un arco
- *Flag***n** de un arco
- *Peso***n** de un arco
- *Capacidad***n:** cambiar las capaciadades
- *HoraDeLlegada*
- *HoraDeAtención*
- *HoraDeSalida*
- *Capacidad_Demanda_Restante***n:** cambiar la demanda o la capacidad N restante.
- Encontrar en una red particular todos los clientes que cumplen una Flag

Algoritmos

Se debe implementar un conjunto de algoritmos sobre la estructura de datos generada. Cada algoritmo debe tomar como "peso" de la RED, el indicado por el usuario.

- Debe permitir "navegar sobre la estructura", o sea, dado un objeto del tipo línea poder saber que conexión tiene con otros objetos y que costo le insume.
- Dado un punto poder encontrar: cual es el punto más cercano, lejano, o el que se encuentre dentro de un rango de costo particular; el par de puntos más próximos a él (hacia atrás y adelante), los n puntos más cercanos. Esto puede ser para un punto de la misma categoría, o de cualquier otra.

ALGORITMOS DE PRUEBA

Deben realizar un conjunto de algoritmos para la prueba de la estructura de datos. Todos los algoritmos se deben resolver usando pura y exclusivamente las estructuras de datos y la interface creada.

El algoritmo de Dijkstra que calcula la distancia mínima de un punto al resto. Este lo deben implementar para poder tener la información de la distancia mínima entre dos puntos, no necesariamente deben usar la interface, pude hacer uso del conocimiento que se tienen de las estructuras de datos creadas por ustedes.

Tsp

Algoritmo Tsp de "*Rosenkrantz, Sterns and Lewis", Nearest Insertion* (pag.88 – L.Bodin).

Asignación

Asignar clientes a depósitos, teniendo en cuenta la demanda de los clientes y las capacidades de los depósitos.

Asignación – Tsp

Conectar el TSP anterior con la Asignación. Sobre cada asignación se realiza un Tsp.

Descripción del Algoritmo de Asignación

Idea general

Para cada cliente no asignado, calcula la suma de: las distancias a cada depósito (aun no agotado, o sea que no excedió la capacidad) menos la distancia a su depósito mas cercano. Luego asigna aquel cliente que maximiza dicha suma, a su depósito mas cercano.

Especificación

Sea:

- C1.. Cn conjunto de clientes a ser asignados
- D1..Dm conjunto de depósitos
- K1..Km capacidad de cada depósito
- Dist(Cj, Dk) distancia del cliente j al depósito k

∀ Cj no asignado y Dk no saturado, se calcula: Dist(Cj, Dk)
sea Mj = Min(∀ k Dist(Cj, Dk)) o sea la distancia a su depósito más cercano.

Asigno aquel cliente tal que Max ((Σ Dist(Cj, Dk) en k) - Mj) a su depósito mas cercano.

El control de la capacidad de los depósitos que se realiza es la siguiente: primero asigno el cliente y luego controlo si el depósito se saturó, por lo tanto puede exceder la capacidad del depósito ya que si tiene una disponibilidad el depósitos de 10 y le asigno un cliente con 15, queda excedido el depósito en 5.

Luego de que un depósito se satura, se desasignan todos los clientes de los depósitos que aún no se encuentran saturados y se vuelve a comenzar con ellos.

Funciones

Aquí se especifican algunas de las funciones que se deben realizar sobre la estructura de datos. *Pueden cambiar los nombre y los parámetros de ser necesario siempre y cuando mantengan un criterio, además las funciones especificadas aquí no son las únicas, en caso de necesitar especificar algunas nuevas está librado a su buen criterio. Sugiero*

SOBRE LA RED

Funciones que se aplican a la red (shapefile de líneas)

CARGARED(ARCHIVO)

Carga la red desde un shapefile de arcos, o archivo Asccii(C++).
Puedo cargar varias

CARGAIMPEDANCIARED(ARCHIVO)

Carga en la red un conjunto de impedancias a los arcos especificados
CantidadDeNodos

Apéndice 2:

ORGANIZACIÓN Y SEGUIMIENTO DEL PROYECTO.

Identificación de Tareas

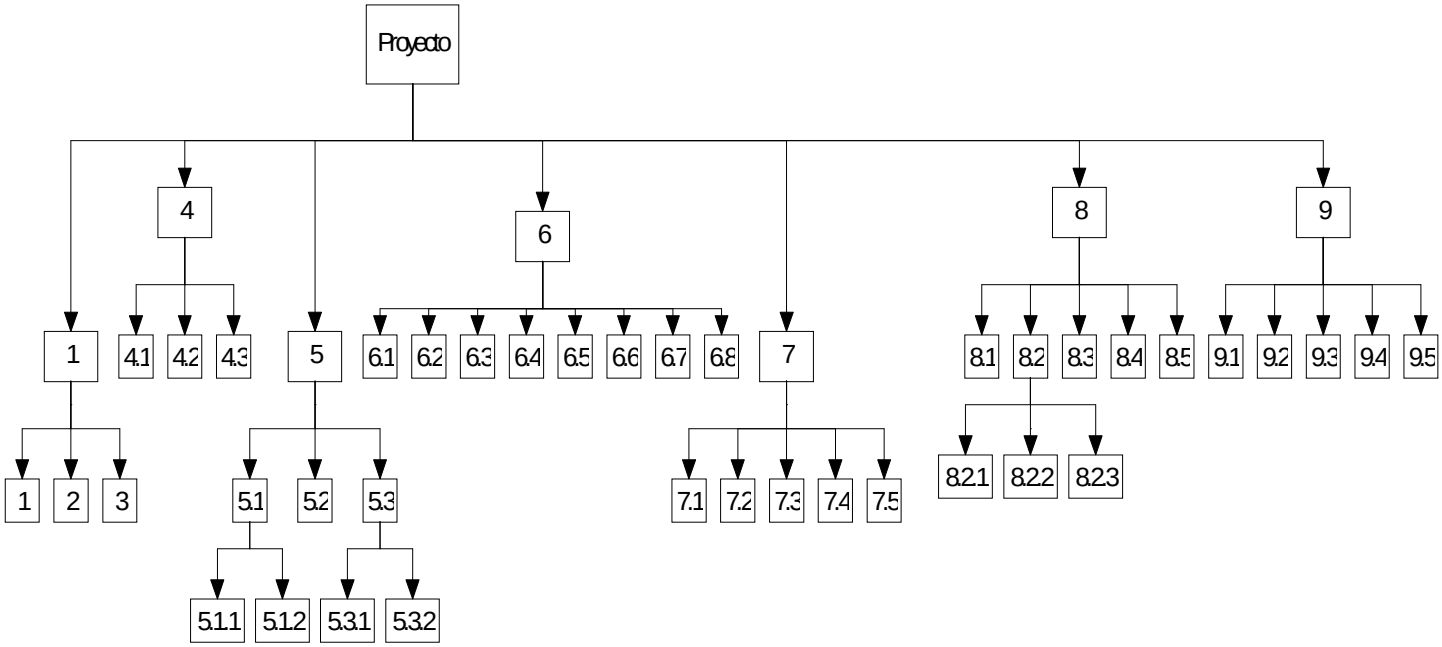
1. Análisis de requerimientos
2. Aprendisaje de las herramientas
3. Análisis de la forma en que las herramientas afectan los requerimientos
4. Diseño
   - 4.1. Alto nivel (Mor)
   - 4.2. Bajo nivel
   - 4.3. Nivel de implementación
5. Investigación
   - 5.1. Estudio del problema
     - 5.1.1. Vehicle Routing: Methods and studies [2]
     - 5.1.2. Routing and scheduling of vehicles and crews. The state of the art [1]
   - 5.2. GIS y Relación con VRP
   - 5.3. Formatos
     - 5.3.1. Shp
     - 5.3.2. dbf
6. Planificación
   - 6.1. Identificación de tareas
   - 6.2. Planificación preliminar
   - 6.3. Gantt
   - 6.4. Planificación Post-Diseño
   - 6.5. PERT
   - 6.6. Gantt
7. Documentación
   - 7.1. Elaboración del estándar
   - 7.2. Elaboración de la Plantilla Word
   - 7.3. Armado del esqueleto de la entrega final
   - 7.4. Elaboración de la entrega final
   - 7.5. Elaboración de la presentación final
8. Implementación
   - 8.1. Elaboración del estándar
   - 8.2. Implementación de Interfaces
     - 8.2.1. Entre shp y archivos de texto
     - 8.2.2. Entre dbf y archivos de texto
     - 8.2.3. Unificación de las interfaces
   - 8.3. Implementación de Módulos
   - 8.4. Implementación de la Estructura final
   - 8.5. Implementación de los algoritmos de Asignación, TSP y ambos combinados sobre la estructura.
9. Testeo
   - 9.1. Elaboración de un plan de Testeo
   - 9.2. Testeo de la interface
   - 9.3. Testeo de módulos

9.4. Testeo de la estructura final
9.5. Testeo de los algoritmos

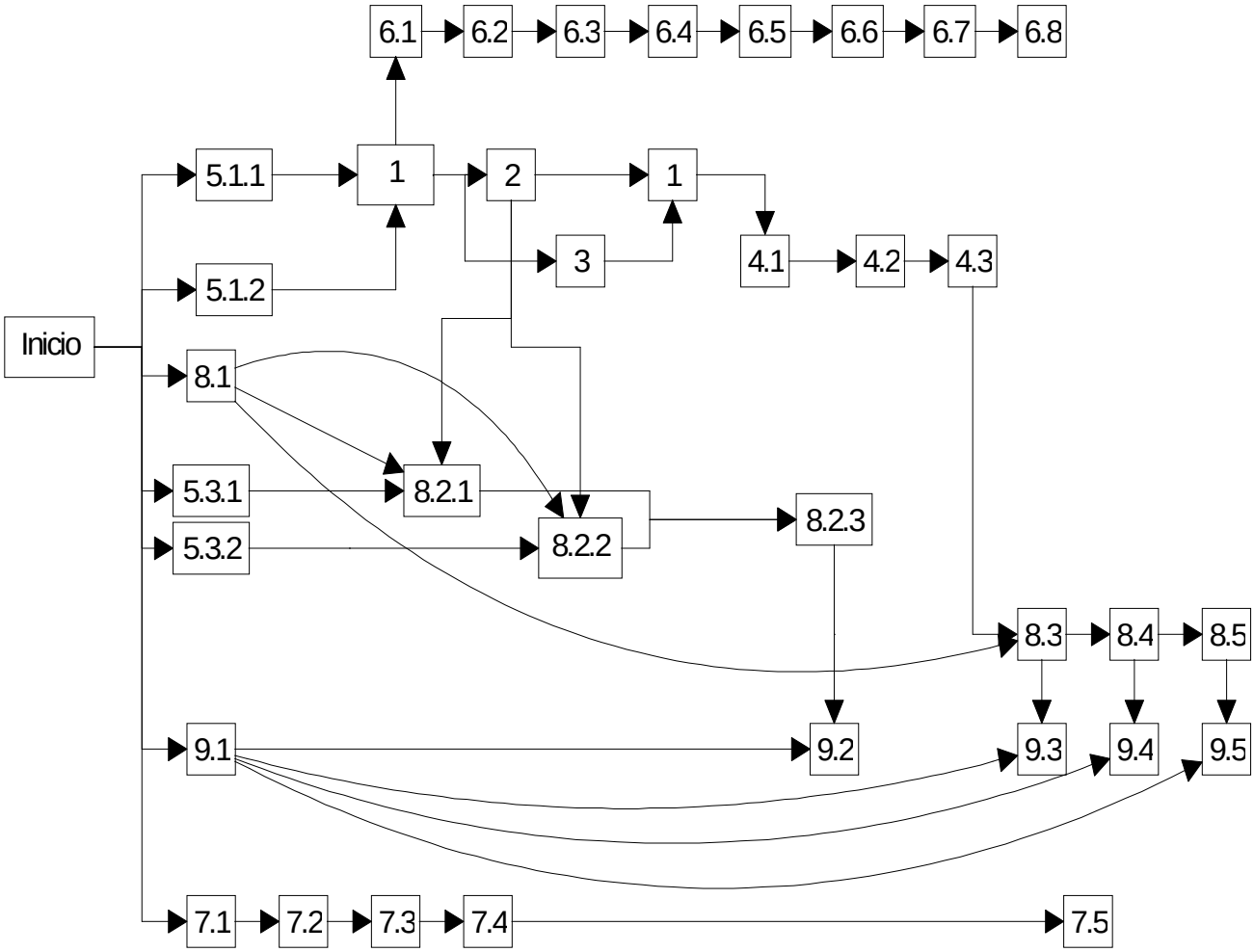
Planificación preliminar

Nombre de tarea	Duración	Comienzo	Fin	Prede-cesoras
Análisis de Requerimientos	16 días	lu 26/04/99	lu 17/05/99	
Aprendizaje de las herramientas	10 días	ma 18/05/99	lu 31/05/99	1
Análisis de la forma en que las herramientas afectan los requerimientos	10 días	ma 01/06/99	lu 14/06/99	2
Diseño de alto nuvel (MOR)	2 días	lu 17/05/99	vi 17/05/99	
Armado del esqueleto de la entrega final	1 día	lu 07/06/99	lu 07/06/99	
Elaboración de estándar de Implementación	5 días	lu 02/06/99	mi 09/06/99	
Elaboración de Estándar de Documentación	5 días	lu 26/05/99	mi 02/06/99	
Elaboración de Plantilla Word para el Estándar de Documentación	5 días	lu 26/05/99	mi 02/06/99	
Elaboración de la entrega final		lu 07/06/99	ma 06/07/99	
Diseño de bajo nivel	15 días	ma 21/07/99	lu 11/08/99	
Implementación de Estructuras genéricas a ser usadas en la estructura a implementar	15 días	ju 01/07/99	vi 15/07/99	
Implementación de la estructura planteada	92 días	lu 15/08/99	vi 19/11/99	12
Armado de la entrega Final	10 días	mi 23/06/99	ma 06/07/99	13
Investigación sobre formato .shp	5 días	ju 20/05/99	mi 26/05/99	
Investigación sobre formato .dbf	10 días	ju 27/05/99	mi 09/06/99	
Implementación de Interface entre formato .shp y archivo de texto	5 días	ju 27/05/99	mi 02/06/99	15
Implementación de Interface entre formato .dbf y archivo de texto	5 días	ju 10/06/99	mi 16/06/99	16
Unificación de las interfaces	5 días	ju 17/06/99	mi 24/06/99	16,15

WBS


Proyecto
1
4
5
6
7
8
9
1
2
3
4.1
4.2
4.3
5.1
5.2
5.3
6.1
6.2
6.3
6.4
6.5
6.6
6.7
6.8
7.1
7.2
7.3
7.4
7.5
8.1
8.2
8.3
8.4
8.5
8.2.1
8.2.2
8.2.3
9.1
9.2
9.3
9.4
9.5
5.1.1
5.1.2
5.3.1
5.3.2

PERT

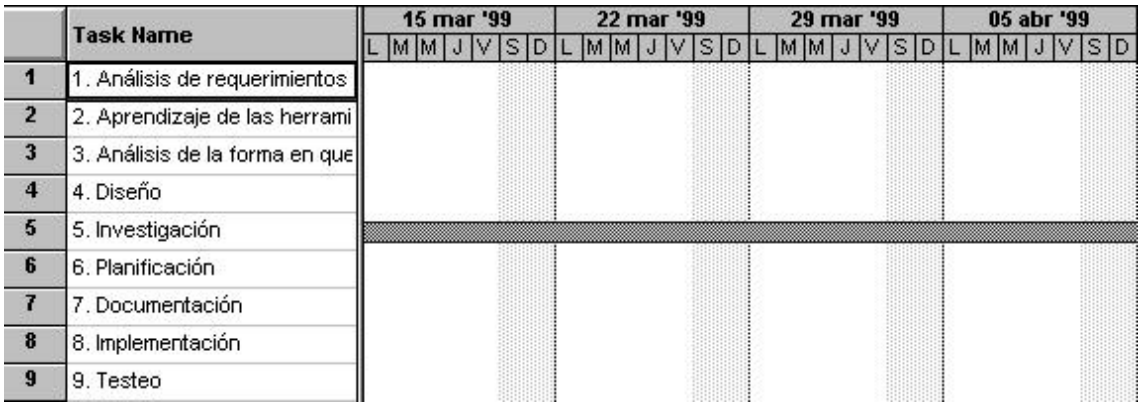

6.1
6.2
6.3
6.4
6.5
6.6
6.7
6.8
5.1.1
1
2
1
3
4.1
4.2
4.3
5.1.2
Inicio
8.1
5.3.1
8.2.1
8.2.3
5.3.2
8.2.2
8.3
8.4
8.5
9.1
9.2
9.3
9.4
9.5
7.1
7.2
7.3
7.4
7.5

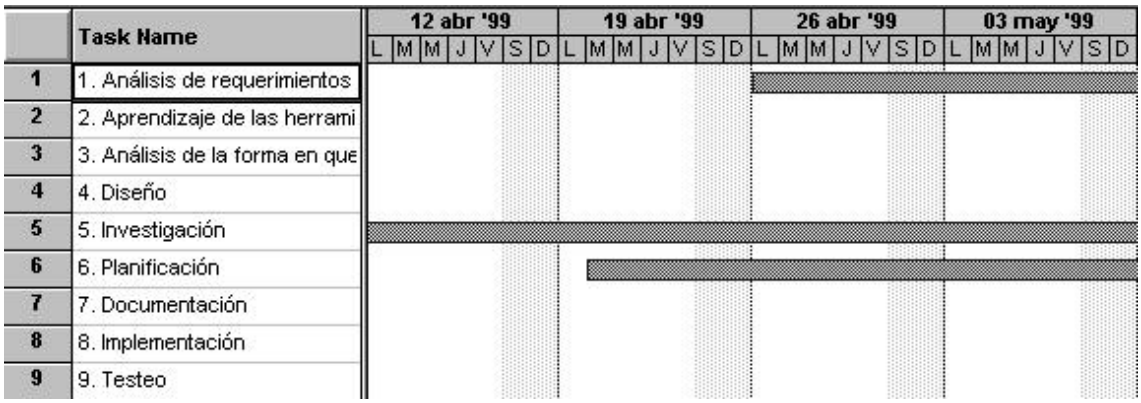
Planificación

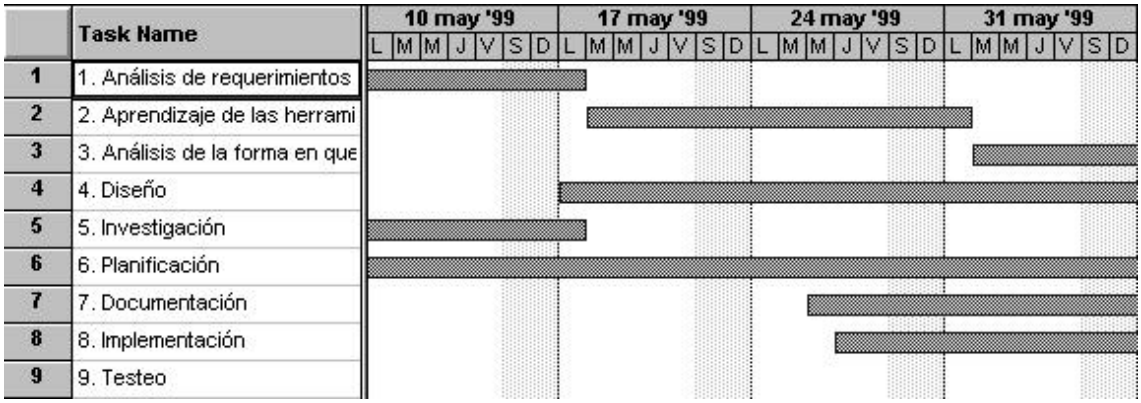
Identificación de Tareas

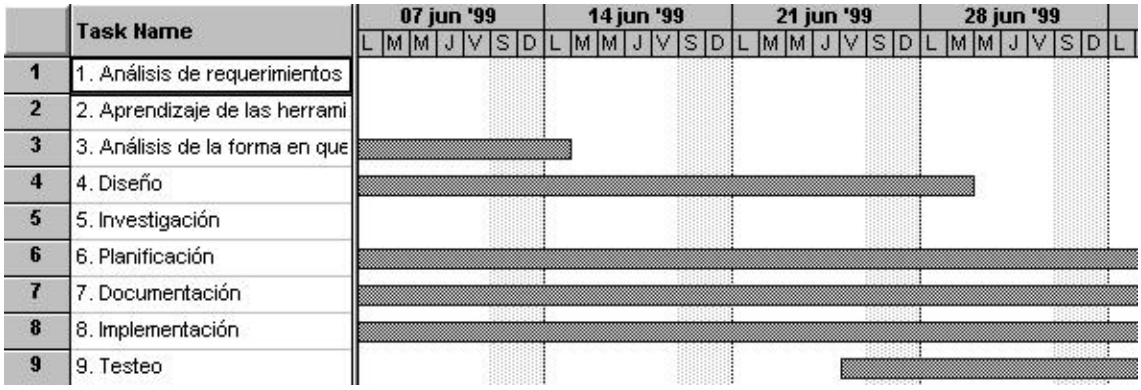
Tarea	**Duración**	**Inicio**	**Fin**
1. Análisis de requerimientos	16	26/4/99	17/5/99
2. Aprendisaje de las herramientas	10	18/5/99	31/5/99
3. Análisis de la forma en que las herramientas afectan los requerimientos	10	1/6/99	14/6/99
4. Diseño	**32**	**17/5/99**	**4/9/99**
4.1. Alto nivel (Mor)	2	17/5/99	19/5/99
4.2. Bajo nivel	15	27/5/99	9/6/99
4.3. Nivel de implementación	15	14/8/99	4/9/99
5. Investigación	**30**	**6/4/99**	**9/6/99**
5.1. Estudio del problema	**10**	**6/4/99**	**13/4/99**
5.1.1. Vehicle Routing: Methods and studies [2]	5	6/4/99	13/4/99
5.1.2. Routing and scheduling of vehicles and crews. The state of the art [1]	5	6/4/99	13/4/99
5.2. GIS y Relación con VRP	5	6/4/99	13/4/99
5.3. Formatos	**15**	**20/5/99**	**9/6/99**
5.3.1. Shp	5	20/5/99	26/5/99
5.3.2. dbf	10	27/5/99	9/6/99
6. Planificación	**9**	**23/6/99**	**15/9/99**
6.1. Identificación de tareas	2	23/6/99	25/6/99
6.2. Planificación preliminar	1	24/6/99	25/6/99
6.3. Gantt	1	25/6/99	26/6/99
6.4. Planificación Post-Diseño	3	6/9/99	9/9/99
6.5. PERT	1	10/9/99	11/9/99
6.6. Gantt	1	14/9/99	15/9/99
7. Documentación	**46**	**26/2/99**	**14/12/99**
7.1. Elaboración del estándar	5	26/5/99	2/6/99
7.2. Elaboración de la Plantilla Word	5	26/5/99	2/6/99
7.3. Armado del esqueleto de la entrega final	1	7/6/99	8/6/99
7.4. Elaboración de la entrega final	30	8/6/99	6/7/99
7.5. Elaboración de la presentación final	5	1/12/99	14/12/99
8. Implementación	**112**	**27/5/99**	**26/11/99**
8.1. Elaboración del estándar	5	2/6/99	9/6/99
8.2. Implementación de Interfaces	**15**	**27/5/99**	**24/6/99**
8.2.1. Entre shp y archivos de texto	5	27/5/99	2/6/99
8.2.2. Entre dbf y archivos de texto	5	10/6/99	16/6/99
8.2.3. Unificación de las interfaces	5	17/6/99	24/6/99
8.3. Implementación de Módulos	74	15/8/99	26/10/99
8.4. Implementación de la Estructura final	18	27/10/99	19/11/99
8.5. Implementación de los algoritmos de Asignación, TSP y ambos combinados sobre la estructura.	5	22/11/99	26/11/99
9. Testeo	**42**	**25/6/99**	**10/12/99**
9.1. Elaboración de un plan de Testeo	2	5/9/99	7/9/99
9.2. Testeo de la interface	5	25/6/99	30/6/99
9.3. Testeo de módulos	24	27/10/99	16/11/99
9.4. Testeo de la estructura final	10	23/11/99	7/12/99
9.5. Testeo de los algoritmos	2	8/12/99	10/12/99

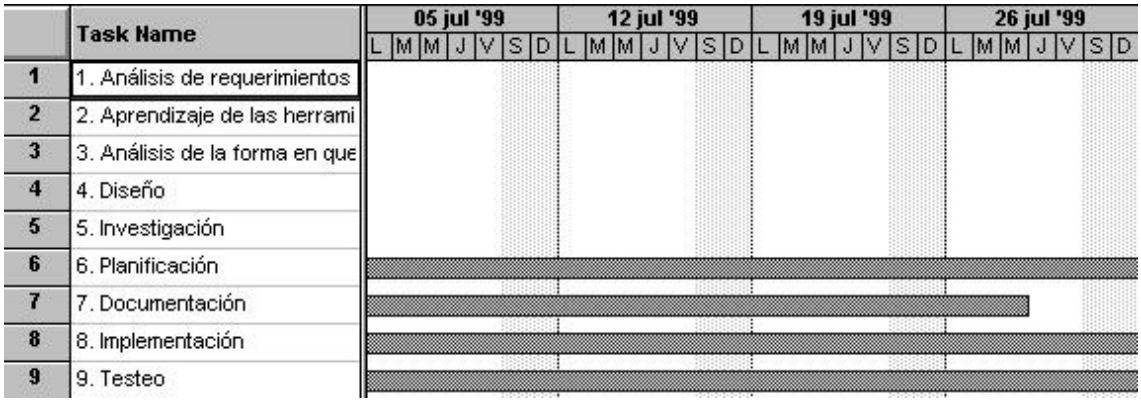
Gantt

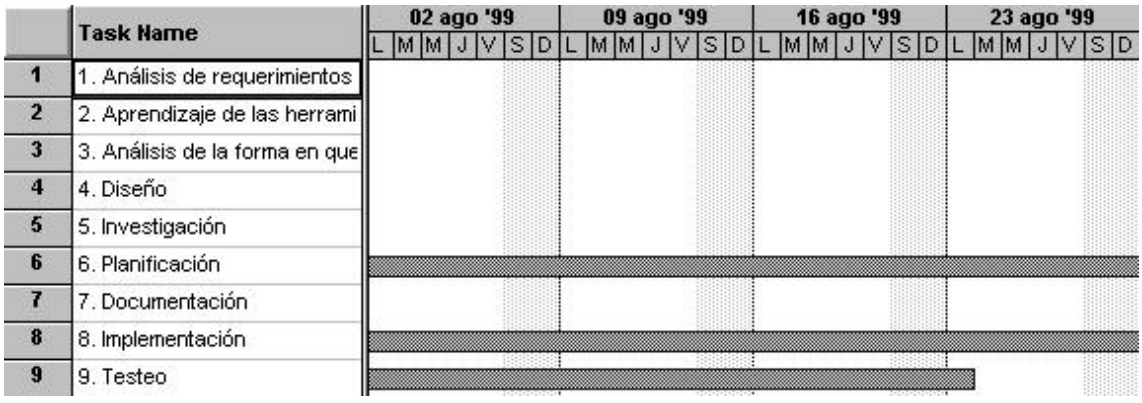

Task Name
15 mar '99
22 mar '99
29 mar '99
05 abr '99
L M M J V S D
1 1. Análisis de requerimientos
2 2. Aprendizaje de las herrami
3 3. Análisis de la forma en que
4 4. Diseño
5 5. Investigación
6 6. Planificación
7 7. Documentación
8 8. Implementación
9 9. Testeo

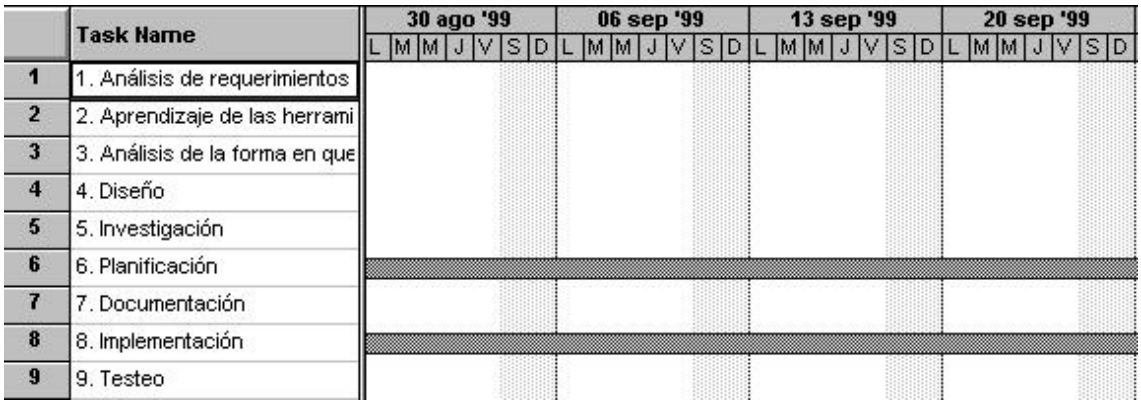

Task Name
12 abr '99
19 abr '99
26 abr '99
03 may '99
L M M J V S D
1 1. Análisis de requerimientos
2 2. Aprendizaje de las herrami
3 3. Análisis de la forma en que
4 4. Diseño
5 5. Investigación
6 6. Planificación
7 7. Documentación
8 8. Implementación
9 9. Testeo

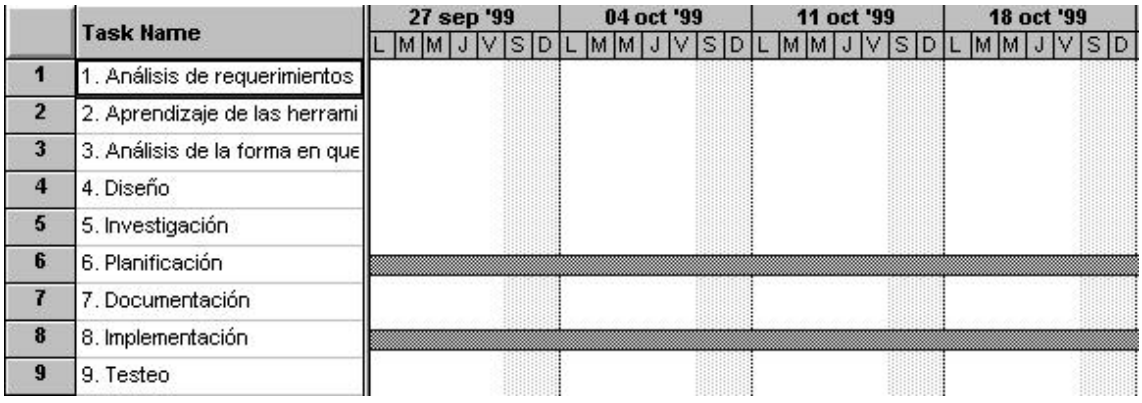

Task Name
10 may '99
17 may '99
24 may '99
31 may '99
L M M J V S D
1 1. Análisis de requerimientos
2 2. Aprendizaje de las herrami
3 3. Análisis de la forma en que
4 4. Diseño
5 5. Investigación
6 6. Planificación
7 7. Documentación
8 8. Implementación
9 9. Testeo

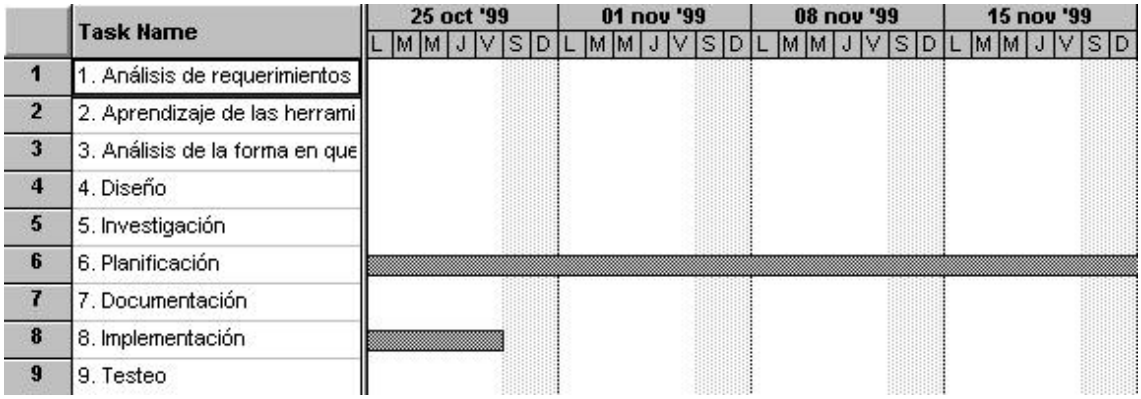

Task Name
07 jun '99
14 jun '99
21 jun '99
28 jun '99
L M M J V S D
1 1. Análisis de requerimientos
2 2. Aprendizaje de las herrami
3 3. Análisis de la forma en que
4 4. Diseño
5 5. Investigación
6 6. Planificación
7 7. Documentación
8 8. Implementación
9 9. Testeo

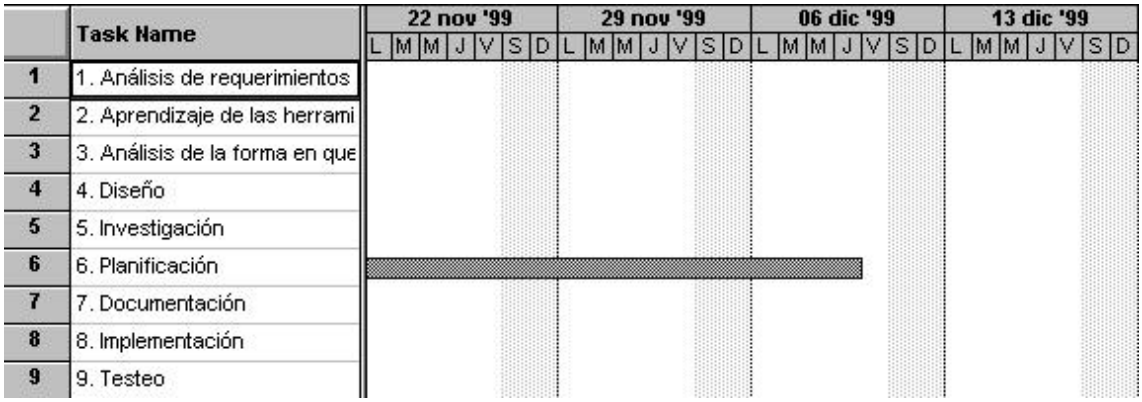

Task Name
05 jul '99
12 jul '99
19 jul '99
26 jul '99
L M M J V S D L M M J V S D L M M J V S D L M M J V S D
1 1. Análisis de requerimientos
2 2. Aprendizaje de las herrami
3 3. Análisis de la forma en que
4 4. Diseño
5 5. Investigación
6 6. Planificación
7 7. Documentación
8 8. Implementación
9 9. Testeo


Task Name
02 ago '99
09 ago '99
16 ago '99
23 ago '99
L M M J V S D L M M J V S D L M M J V S D L M M J V S D
1 1. Análisis de requerimientos
2 2. Aprendizaje de las herrami
3 3. Análisis de la forma en que
4 4. Diseño
5 5. Investigación
6 6. Planificación
7 7. Documentación
8 8. Implementación
9 9. Testeo


Task Name
30 ago '99
06 sep '99
13 sep '99
20 sep '99
L M M J V S D L M M J V S D L M M J V S D L M M J V S D
1 1. Análisis de requerimientos
2 2. Aprendizaje de las herrami
3 3. Análisis de la forma en que
4 4. Diseño
5 5. Investigación
6 6. Planificación
7 7. Documentación
8 8. Implementación
9 9. Testeo


Task Name
27 sep '99
04 oct '99
11 oct '99
18 oct '99
L M M J V S D L M M J V S D L M M J V S D L M M J V S D
1 1. Análisis de requerimientos
2 2. Aprendizaje de las herrami
3 3. Análisis de la forma en que
4 4. Diseño
5 5. Investigación
6 6. Planificación
7 7. Documentación
8 8. Implementación
9 9. Testeo


Task Name
25 oct '99
01 nov '99
08 nov '99
15 nov '99
L M M J V S D L M M J V S D L M M J V S D L M M J V S D
1 1. Análisis de requerimientos
2 2. Aprendizaje de las herrami
3 3. Análisis de la forma en que
4 4. Diseño
5 5. Investigación
6 6. Planificación
7 7. Documentación
8 8. Implementación
9 9. Testeo


Task Name
22 nov '99
29 nov '99
06 dic '99
13 dic '99
L M M J V S D L M M J V S D L M M J V S D L M M J V S D
1 1. Análisis de requerimientos
2 2. Aprendizaje de las herrami
3 3. Análisis de la forma en que
4 4. Diseño
5 5. Investigación
6 6. Planificación
7 7. Documentación
8 8. Implementación
9 9. Testeo

Apéndice 3:

NORMAS Y ESTÁNDARES DE IMPLEMENTACIÓN

Alcance

El presente capítulo describe las normas y estándares de implementación, es una guía con el fin de obtener un código más legible y facilitar la interpretación del código.

Contenido

Normas Generales de Programación.

Variables

Las variables deben tener siempre el menor alcance posible. Se debe evitar a toda costa el uso de variables globales, ya que la utilización de variables globales no es una buena práctica de programación, ya que extienden el área de modificación de la variable, haciendo con esto muy tediosa y difícil la tarea de detectar posibles fallas en nuestros programas. En caso de no poder evitarse su utilización se deberá comentar esta decisión.

El identificador de una variable debe ser mnemotécnico, aunque en este sentido primará la flexibilidad y sensatez permitiéndose la utilización identificadores como **I** y **J** para índices de arreglos o **N** para el número de elementos de un conjunto por su uso común en la matemática; sin embargo identificadores como **FILA**, **COLUMNA**, y **CANTIDAD** resultarán mejores. En aquellos casos donde el identificador de una variable no sea lo suficientemente claro como para indicar su semántica se deberá agregar un comentario que clarifique su utilización.

En caso de que la variable sea global, el nombre de la misma comenzará con _, por ej.: **_VARGLOBAL.**

Constantes

Los nombres de las constantes pueden contener únicamente mayúsculas y _, en caso de requerirse para mayor claridad, por ejemplo: **CONST_ENTERA**. Deben ser mnemotécnicas, es decir su nombre deberá, en la medida de lo posible, indicarnos su contenido.

Clases

La utilización de clases es una de las herramientas más potentes para lograr la modularidad en la programación.

A la hora de escoger el nombre de una clase el mismo deberá ser mnemotécnico, sólo podrá contener caracteres en minuscula y deberá comenzar con c_, por ejemplo: **C_NODO.**

El nombre de las variables privadas de la clase deberá comenzar con **M_**, podrá contener únicamente caracteres en mínuscula, por ejemplo: **INT M_DATOCLASE.**

Indentación

Una apropiada y consistente indentación es requerida para enfatizar la estructura de un programa.

La utilización de indentación será obligatoria.

Comentarios

Los comentarios deben brindar una visión global del código y/o explicar algunas partes claves, evitándose especialmente comentarios extensos.

El comentario de una rutina deberá indicar lo que la rutina hace, no cómo lo hace.

Las variables deben llevar un pequeño comentario de lo que hacen, si su identificador no es mnemotécnico.

Los comentarios de las funciones y procedimientos deben ubicarse en la siguiente línea, luego de la de los mismos.

Formato del Código

Se debe seguir el siguiente orden en las declaraciones:

1. Constantes
2. Tipos
3. Variables
4. Cuerpo de la rutina.

Dentro del código no pueden aparecer declaraciones de variables, constantes o tipos, todas las declaraciones deben ser hechas al principio.

Normas de Programación en C++.

Variables

Para los nombres de variables se utilizará un estilo en el que se combinan mayúsculas y minúsculas buscando mayor claridad a la hora de escoger el nombre de una variable, por ejemplo se utilizarán nombres con el siguiente formato: **CotaSuperior, CotaInferior**, etc. Los nombres pueden comenzar con mayúscula o minúscula, pero deben contener al menos un carácter en minúscula, no son válidos identificadores que utilicen únicamente caracteres en mayúscula. Solo se pueden utilizar caracteres alfanuméricos en el nombre de una variable.

Tomamos como regla NO inicializar las variables en el momento de su declaración.

Constantes

Para declarar constantes utilizaremos la sentencia **CONST**, que especifica que el valor de una variable es constante, impidiendo que el valor de la misma sea modificado.

Análogamente se podría utilizar la directiva para el compilador **#DEFINE**, pero las constantes definidas de esta manera no serían objeto del chequeo de tipos realizado por el compilador.

Ejemplo: **Const int TOPE_SUPERIOR = 5;**

Clases

Para todas las clases sin excepción se deberá implementar como mínimo el constructor sin parámetros y el destructor, evitando de esta forma la utilización del que por defecto declara el compilador.

Se prohibe la declaración de rutinas **FRIEND** en una clase. Recordemos que las rutinas declaradas como **FRIEND**, no son miembros de la clase, sino que son rutinas que se declaran dentro de una clase y que tienen acceso a la parte privada de la misma.

La estructura de nuestra clase deberá siempre ser **PRIVATE**, y bajo ningún concepto se permitirá su definición como **PUBLIC** o **PROTECTED**.

Para nombrar los métodos de una clase se utilizará una combinación de mayúsculas y minúsculas, buscando resaltar el nombre de los mismos, haciendo que cada palabra que compone el nombre del método comience con mayúscula, por ejemplo para dar nombre a una rutina que halla el mejor camino entre dos puntos utilizaremos: **MEJORCAMINO (...)**.

Se restringirá la definición de funciones **INLINE**, permitiéndose la utilización de las mismas únicamente en casos debidamente justificados. Será práctica habitual que la definición de las clases este en los **.H** y la implementación de las mismas en forma integra en los **.CPP**.

Rutinas (procedimientos y funciones)

Para nombrar las Rutinas se utilizará una combinación de mayúsculas y minúsculas, buscando resaltar el nombre de los mismos, haciendo que cada palabra que compone el nombre del método comience con mayúscula, por ejemplo para dar nombre a una rutina que halla el mejor camino entre dos puntos utilizaremos: **MEJORCAMINO (...)**.

Obligatoriamente todas las rutinas deben ser comentadas, el comentario de las mismas se comenzará en la línea siguiente a su definición.

Indentación

Aprovechando el hecho de que la herramienta de programación Visual C++ 6.0, indenta automáticamente nuestro código, lo único que resta por hacer es definir la cantidad de caracteres que se utilizan para indentar. Tomaremos como estándar una indentación de 4 caracteres.

Comentarios

Deben estar al mismo nivel de indentación que el código que describen.
Para comentarios en una línea:

//COMENTARIO

Para comentarios en múltiples líneas:

/* LÍNEA 1 DEL COMENTARIO
LÍNEA 2 DEL COMENTARIO
.....
LÍNEA N DEL COMENTARIO */

La semántica de los miembros de la clase será comentada en el **.H**, indicando brevemente en cada caso el propósito de dicho miembro.

Formato del Código

Existen en C++, un conjunto de sentencias que si bien resultan especialmente útiles en algunas ocasiones tienden a oscurecer la codificación, haciendo a nuestros programas de difícil lectura y comprensión, a continuación especificamos estas sentencias, y la forma en la que utilizaremos cada una de las mismas:

SENTENCIA GOTO

No se utilizará la sentencia goto bajo ningún concepto y forma.

SENTENCIA BREAK

Se evitará su utilización para salir de iteraciones, utilizándose la misma únicamente en conjunto con la sentencia switch.

SENTENCIA RETURN

Será deseable que la sentencia **RETURN** aparezca únicamente, al final del código de una función, indicando el resultado de la misma, evitándose especialmente salir de iteraciones utilizando esta sentencia.

Apéndice 4:

SISTEMAS DE INFORMACIÓN GEOGRÁFICA (GIS)

Los Sistemas de Información Geográfica (GIS), pueden ser usados para investigaciones científicas, administración de recursos, simulación de eventos, planificación de desarrollo, así como otros

¿ Qué es un GIS ?

Se han dado muchas definiciones de lo que es un GIS pero en un sentido estricto podemos decir que un GIS en un sistema informático capaz de reunir, almacenar, manipular y visualizar información geográficamente referenciada, es decir datos identificados por su ubicación en un mapa.

¿ De qué forma trabaja un GIS ?

- RELACIONANDO INFORMACIÓN DE DISTINTOS ORÍGENES.

Por lo general un GIS tomará un mapa como base de trabajo sobre la cuál almacenará y desplegará datos. Es común que el mapa y los datos no provengan siempre de la misma fuente, será aquí en donde el GIS deberá relacionar las distintas fuentes de información para unificar todo dentro del mapa, por ejemplo podrían vincularse fotos áreas de un satélite con los porcentajes de lluvia.

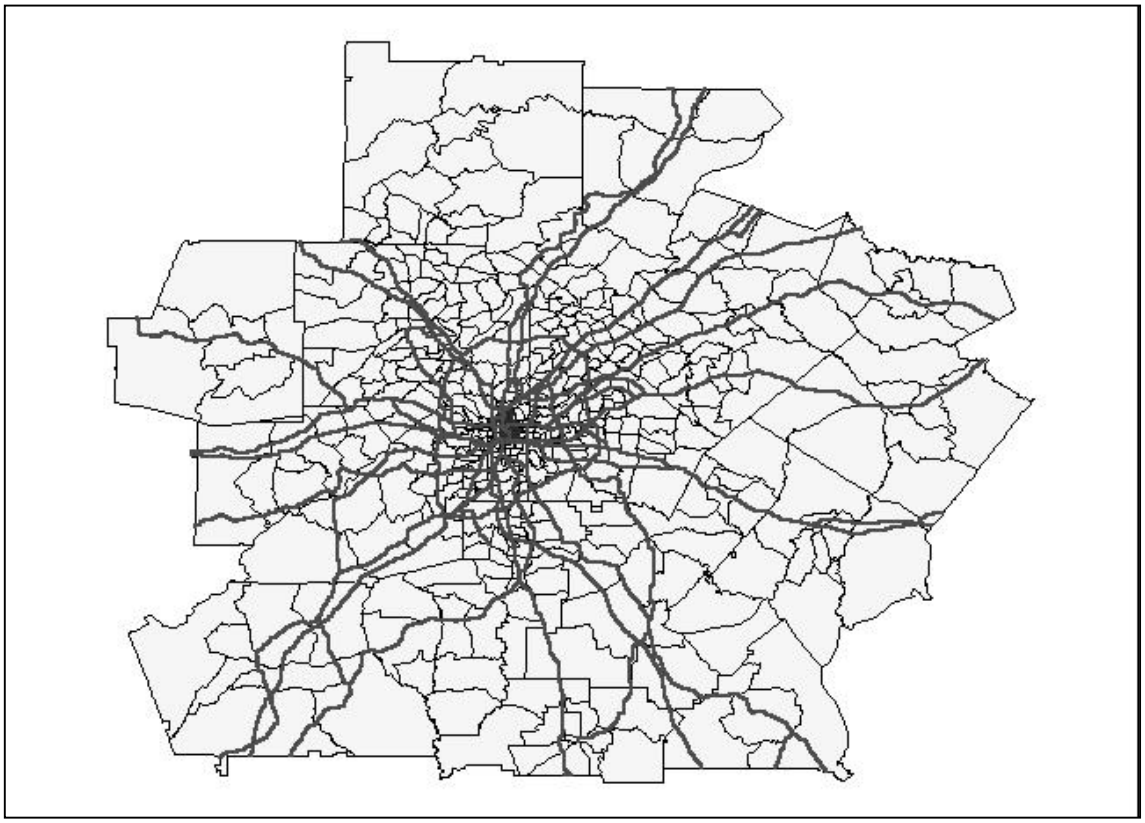


Figura 1. Mapa de Atlanta que viene con Arcview

- HACIENDO CAPTURA DE DATOS.

Si la información a ser utilizada no está ya digitalizada, la misma debe ser convertida a un formato en el que la misma pueda ser reconocida por una computadora, los mapas pueden ser digitalizados, o dibujados utilizando programas destinados a estos efectos y de esta forma capturar las coordenadas de los distintos elementos.
Un GIS puede ser utilizado para enfatizar la relación espacial que existe entre los distintos elementos de un mapa, por ejemplo mientras que en un mapa digital una calle será seguramente representada y vista como una línea, en un GIS podrá reconocerse esa calle como el borde entre un área de bosques y un área urbanizada, o porque no como el enlace que une dos ciudades.
El proceso de la captura de datos, es por lo general una de las etapas que consume mayor cantidad de tiempo dentro de la ardua tarea que consiste en la creación de un GIS.

- HACIENDO INTEGRACIÓN DE LOS DATOS.

Un GIS hace posible la vinculación o integración de información que de otra manera resultaría muy difícil de vincular. Por ejemplo con la utilización de un GIS una compañía de agua podrá determinar que cantidad de material séptico deberá destinar para la potabilización del agua consumida en una determinada área, esto se hará vinculando a un GIS el consumo de agua de una determinada zona.

- DEFINIENDO ESTRUCTURAS DE DATOS.

Cómo ya mencionamos una de las principales propiedades de un GIS es que permite vincular información que de otra forma resultaría prácticamente imposible vincular. Para poder hacer esta vinculación un GIS deberá tener definidas las estructuras de datos necesarias para poder efectuar está tarea, por ejemplo será necesario identificar y vincular distintos puntos del mapa digital como pertenecientes a una misma zona, por lo cual son necesarias estructuras de datos que permitan hacer esta vinculación.

- MODELADO DE DATOS.

Una de las principales características de un GIS se encuentra vinculada al hecho de que el poder almacenar información relacionada con un mapa digital permite representar la idea tridimensional de la Tierra, por ejemplo a través de un GIS podemos conocer los contornos de las zonas con igual índices de lluvias, o con igual índices de temperaturas, es decir podemos desplegar distintos tipos de información basados en el mismo mapa base, de acuerdo a la información que hayamos vinculado al mismo.

¿ Cuál es la función de un GIS ?

La forma en qué los datos son almacenados y desplegados permiten que un GIS sea una herramienta que resulta especialmente adecuada para la realización de complejos análisis.

- DEVOLUCIÓN DE INFORMACIÓN.

La manera típica en que un GIS devuelve información es la siguiente, cuando una persona cliquea sobre un punto específico del mapa la información almacenada relacionada ese punto es desplegada, también se podría pedir que coloreando el mapa el

GIS pinte de un mismo color todas aquellas áreas en donde variables específicas tienen el mismo valor.

- MODELO TOPOLÓGICO.

A través de un GIS pueden manejarse relaciones de espaciales entre los distintos objetos que conforman nuestro sistema, por lo que un GIS nos permitirá determinar condiciones de adyacencia, proximidad, inclusión, entre los distintos objetos que forman parte del mapa digital.

- REDES.

Un GIS permite almacenar información referente a distintas redes, como por ejemplo una red de distribución, de esta forma se pueden simular distintos eventos como por ejemplo el cálculo del tiempo total que llevaría un recorrido en la antes mencionada red de distribución.

- SALIDA.

Por lo general la salida de un GIS resultará un gráfico en pantalla, que permitirá el análisis de las personas que deben tomar decisiones de la situación existente reflejada en ese mapa. Es entonces un aspecto crítico de un GIS la forma en que las respuestas son mostradas al usuario, teniendo en cuenta que información será más relevante para el usuario, dejando de lado aquella que no aporte nada en el caso específico que se esta tratando.

Aplicaciones de GIS

- GIS A TRAVÉS DE LA HISTORIA.

En las paredes de las cavernas de Lascaux, Francia, el hombre de Cro-Magnon dibujo imágenes de los animales que cazaba, asociado a estos dibujos hay historias que relatan la migración de las especies en las distintas épocas del año. En esencia estos dibujos con sus respectivas historias siguen los dos aspectos fundamentales de un GIS, un archivo gráfico vinculado con objetos de una base de datos; en este caso el dibujo de cada animal vinculado a su ciclo migratorio.

- CREACIÓN DE MAPAS.

Actualmente los investigadores están trabajando para incorporar la experiencia de los cartógrafos en área de GIS, para la producción automática de mapas.

- SELECCIÓN DE LUGARES ESTRÁTEGICOS.

Una de las más recientes aplicaciones de GIS se encuentra en el área de toma de decisiones, en particular en el campo de la selección de lugares estratégicos, por ejemplo volvamos a nuestra compañía de agua, ahora quiere establecer una nueva planta potabilizadora, es en caso en donde la información almacenada en el GIS le permitirá a nuestra empresa determinar la locación de la nueva planta con respecto a las distintas áreas de consumo buscando disminuir los gastos de distribución del agua.

- PLANIFICACIÓN DE SISTEMAS DE EMERGENCIA.

Por ejemplo un GIS puede ser utilizado para combinar la red vial, con información geológica, para analizar en caso de un terremoto el tiempo de respuesta de las unidades de emergencia.

Para poder llevar a cabo está planificación información especifica del problema deberá ser almacenada en nuestro GIS.

- SIMULACIÓN DE EFECTOS CLIMÁTICOS Y DESASTRES AMBIENTALES.

Conociendo características especiales de los terrenos y redes fluviales que atraviesan los mismos, es que se puede simular el efecto de un determinado cambio climático o desastre ambiental, por ejemplo conociendo el modelo de mareas de un río, es que se puede simular y de esta forma prever el efecto de un derramamiento de petróleo en un lugar específico.

- EL FUTURO DE LOS GIS.

Muchas disciplinas se han visto beneficiadas por la aplicación de las técnicas de GIS, pero aún existe una amplia gama de problemas en donde aún no se ha incorporada esta tecnología, pero sería de gran importancia su incorporación, además a los ya existentes GIS, se encuentre actualmente en estudio la posibilidad de expandirlos agregando nuevas características a los mismos.

Dos nuevos campos en donde se está buscando la incorporación de las tecnologías de GIS son los siguientes:

a) HISTORIA Y CAMBIO GLOBAL DEL CLIMA.

La comunidad científica internacional ha reconocido y admitido las consecuencias ambientales de la actividad humana, es en este campo que la tecnología de GIS ha cobrado importante relevancia, como una herramienta de análisis, para comprender este proceso global de cambio. Un variado número de mapas e imágenes de satélites pueden ser combinados para simular la interacción de variados y complejos sistemas naturales.

b) INCORPORANDO EL TIEMPO.

La tecnología de GIS da a los investigadores la habilidad de examinar las variaciones de la Tierra con el correr del tiempo, para ello debe incorporarse a los GIS tradicionales la posibilidad de vincular a cada punto no un único registro, sino varios registros idénticos cuya variable independiente será la fecha u hora en cual dicha información fue vinculada con el punto del mapa.

Conclusiones.

Los GIS tienen gran campo de aplicación, y su principal ventaja sobre la cartografía y otros métodos tradicionales se basa en su característica dinámica, lo que le permite adaptarse con gran facilidad a los cambios topológicos de las áreas en estudio.
El campo de aplicación de los GIS no tiene límites, y puede ser tanto en el área de las investigaciones, como en el área comercial.