CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA
SECCIÓN COMPUTACIÓN

Técnicas de Geometría Computacional para la Construcción del Núcleo de un Sistema de Información Geográfica

Tesis que presenta

Eduardo Vázquez Fernández

Para obtener el grado de

Maestro en Ciencias

En la especialidad de

Ingeniería Eléctrica
(Opción Computación)

Director de la tesis: Dr. Feliú Davino Sagols Troncoso

La tesis fue apoyada por los proyectos No. 29275E y No. I35710-A del CONACyT

México, D.F.

Septiembre, 2001

Agradecimientos

A la Sección de Computación del Centro de Investigación y de Estudios Avanzados del IPN que con su infraestructura me ha permitido llevar a cabo el presente trabajo.

Al Dr. Feliú Davino Sagols Troncoso por su paciencia, enseñanzas y por la dirección que ha hecho de este trabajo.

A los Drs. Guillermo Morales Luna, Juan Manuel Navarro Pineda y Luis Gerardo de la Fraga por el tiempo dedicado en la revisión del escrito de tesis, por las correcciones, sugerencias e ideas hechas al mismo.

Al CONACyT que con los apoyos brindados en los proyectos No. 29275E y No. I35710-A me permitierón finalizar el presente trabajo.

A Mirna Vida Olascoaga por el apoyo incondicional siempre brindado.

A mis queridos sobrinos, Milton Alexis Ortega Vázquez y Linda Ortega Vázquez, que con su encanto me motivan a realizar mejor mi trabajo.

A mis hermanos Alejandra, Lissette, Fidel e Isaac por su cariño y comprensión.

Finalmente, a mi papá, mamá y tía Loren que con su cariño y apoyo moral han hecho posible este trabajo.

RESUMEN

En este trabajo abordamos algunos problemas formales y computacionales relacionados con la representación, manipulación y consulta de información espacial en los mapas de un Sistema de Información Geográfica. Se ha hecho un estudio del estado del arte de la problemática involucrada. Se detectó y demostró la existencia de serios problemas de inestabilidad numérica en el algoritmo de Bentley/Ottmann [5] para encontrar las posibles intersecciones en un conjunto dado de segmentos rectilíneos sobre el plano. Ante la imposibilidad de corregir el método de Bentley/Ottmann se hizo un estudio en la literatura para encontrar un método alternativo, lo que nos llevo al algoritmo de Edelsbrunner para encontrar las intersecciones posibles en un conjunto de rectángulos que finalmente pudimos adaptar para resolver el problema de las intersecciones de un conjunto de segmentos rectilíneos de manera óptima. Se creó un modelo de navegación en mapas a partir de un esquema de cursores basado en la representación de mapas combinatorios de Tutte [34]. Se instrumentó el método "slab" [29] para encontrar la región que contiene a un punto dado en un mapa. Finalmente, se estableció el diseño general de un conjunto mínimo de módulos para instrumentar Sistemas de Información Geográficos que hemos denominado "Núcleo de un Sistema de Información Geográfica". Adicionalmente, se construyeron herramientas de cómputo de propósito general para decodificar mapas en formato *DXF* ("Drawing Interchange File" introducido por AutoCAD [4]).

Índice General

Índice de Figuras

Índice de Tablas

Introducción

Los mapas de un Sistema de Información Geográfica (SIG), contienen esencialmente líneas poligonales y puntos distribuidos en capas que definen regiones, contornos y posiciones puntuales. A estas entidades gráficas (información espacial) se les puede asociar bloques de información nominal, esto es, información referente a características no espaciales que fácilmente puede ser almacenadas, procesadas y consultadas en un Sistema Manejador de Bases de Datos tradicional.

Existe una gran variedad de software disponible de manera gratuita para representar la información nominal de un mapa, no así para representar la información espacial, ya que el desarrollo de la infraestructura requerida es complejo y de costo elevado.

En esta tesis presentamos el diseño general (a nivel de bloques funcionales con descripción de sus interfaces de comunicación) de un *Núcleo de Sistema de Información Geográfica*, que corresponde a lo que, a nuestro juicio, es el conjunto mínimo de componentes para construir Sistemas de Información Geográfica. Dentro del conjunto de módulos que aparecen en nuestro diseño, hemos instrumentado aquellos que tienen que ver con el manejo de la información espacial, dejando bien establecidas las interfaces de comunicación con los demás módulos.

La esencia del esquema de representación, manipulación y consulta de información espacial utilizado está dada por tres algoritmos fundamentales:

1. Para encontrar la intersección de segmentos rectilíneos (ver [5] y Capítulo 3).

2. Para representar mapas internamente (ver [27] y Capítulo 5).

3. Para localizar la región de un mapa que contiene a un punto dado (ver [29] y Capítulo 6).

El primer algoritmo encuentra las intersecciones de un conjunto dado de segmentos rectilíneos sobre el plano de manera óptima. Inicialmente, se instrumenta el método de Bentley/Ottmann [5] para resolver este problema y se demuestra formalmente (Capítulo 3) que presenta problemas de inestabilidad numérica que lo convierten en un método impráctico. Posteriormente, se instrumenta el método de Edelsbrunner (Capítulo 4) que permite encontrar las intersecciones de un conjunto dado de rectángulos sobre el plano sin problemas de inestabilidad numérica y se adapta para resolver el problema de las intersecciones de segmentos de línea.

El segundo algoritmo ofrece una instrumentación cuya generalidad permite extender la representación de mapas a cualquier superficie orientable (como toro, doble toro, etc., no sólo el plano), lo cual permite que los programas desarrollados puedan ser utilizados en otros contextos relacionados con el uso de mapas como son: el diseño de circuitos VLSI [22], la

simplificación de redes por medio de reducciones $\Delta - Y$ [14], así como, en estudios relacionados con la Teoría Topológica de las Gráficas [17]. Si bien el método ha sido tomado de la literatura, en este trabajo construimos un esquema de navegación original basado en el uso de cursores (Sección 5.4).

El tercer algoritmo instrumenta una de las soluciones óptimas registradas en la literatura, que por su complejidad ha resultado un reto de programación interesante. Hasta donde investigamos, no existe ninguna instrumentación de acceso público de este método.

De manera complementaria, dada la necesidad de contar con mapas reales para alimentar nuestros algoritmos, analizamos diferentes sistemas comerciales disponibles para producir mapas, encontramos que en su mayoría estos sistemas permiten exportar sus formatos propietario al formato *DXF* ("Drawing Interchange File" introducido inicialmente por AutoCAD [4]). Finalmente, adoptamos *DXF* (versión 14) en nuestros desarrollos y construimos herramientas de cómputo para extraer información espacial y nominal de mapas en este formato.

De esta forma, esta tesis ofrece una instrumentación eficiente, original, validada y disponible en forma gratuita de algunos algoritmos básicos que permiten representar, manipular y consultar la información espacial de un mapa. Todo esto contemplando, dentro del contexto de la Ingeniería de Software, cómo enlazar la acción de estos algoritmos con los demás componentes de un Sistema de Información Geográfica.

El trabajo esta organizado de la siguiente manera:

En el Capítulo 1, proponemos el diseño general del núcleo de un Sistema de Información Geográfica. Se presenta el diagrama de bloques con los módulos del sistema y se indica cuáles de estos módulos son los que se han instrumentado.

En el Capítulo 2, presentamos, de manera general, las características del formato *DXF* ("Drawing Interchange File" [4]) y explicamos el método para extraer la información espacio-nominal de un mapa en este formato.

En el Capítulo 3, se presenta el método de Bentley/Ottmann [5] para determinar las posibles intersecciones en un conjunto dado de segmentos rectilíneos. Se discute la optimalidad del método y se demuestra que presenta problemas de inestabilidad numérica.

En el Capítulo 4, se describe el método de Edelsbrunner [28] para detectar pares de intersecciones entre rectángulos y se modifica para proporcionar un método, numéricamente estable, que resuelve el problema de las intersecciones de segmentos de línea en el plano.

En el Capítulo 5, se presenta la estructura de datos LADC (Lista de Aristas Doblemente Conectadas) [27] para representar mapas internamente en la computadora y se muestra un modelo de navegación en los mapas a partir de la idea de mapas combinatorios (Tutte [34]).

En el Capítulo 6, se presenta el método "slab" [29] que permite localizar la región que contiene a un punto dado en un mapa.

En el Capítulo 7, se presenta la metodología usada en la verificación y validación de los algoritmos instrumentados (ver [36, 37]).

Capítulo 1

Diseño del núcleo de un SIG

Un Sistema de Información Geográfica (SIG) está integrado por módulos para la captura, limpieza de la información, edición, mantenimiento, consulta y producción de resultados (ver [8], [12] y [11]). Cada uno de estos módulos provee por sí mismo elementos para interconectarse con los demás módulos utilizando las facilidades del sistema operativo que lo soporte; fundamentalmente se sigue una política de colaboración cliente-servidor; así, es posible establecer los vínculos entre la información nominal y espacial, manejar estructuras internas de representación de mapas adecuadas para realizar búsquedas espacio-nominales, establecer el contacto con el servidor de bases de datos y proporcionar el soporte para los procesos servidores que atienden a las aplicaciones finales. En su conjunto, estos módulos con sus protocolos de intercomunicación es lo que nosotros denominamos *Núcleo de un Sistema Manejador de Sistemas de Información Geográfica* o simplemente *Núcleo*. En este Capítulo proponemos una arquitectura particular para este tipo de núcleo.

1.1 Arquitectura general del núcleo propuesto

La arquitectura de núcleo que aquí proponemos, está basada en las facilidades de intercomunicación de procesos mediante sus entradas y salidas estándar, y la posibilidad de establecer diálogos cliente-servidor entre procesos. En la actualidad, prácticamente cualquier sistema operativo ofrece estas facilidades; sin embargo, considerando la amplia difusión que tiene UNIX haremos referencias explícitas a este sistema operativo sin que esto afecte de manera sensible la generalidad de nuestro diseño.

En la Figura 1.1 mostramos la arquitectura general del núcleo que proponemos; en el diagrama, los rectángulos representan procesos independientes o módulos que se enlazan utilizando las facilidades de intercomunicación de procesos de UNIX, las aristas dirigidas (dibujadas con línea contínua) representan flujos de información a través de las entradas y salidas estándar de procesos, mientras que las aristas dirigidas (dibujadas con línea punteada) representan los diálogos cliente-servidor (orientadas del cliente al servidor).

La entrada del núcleo son mapas en formato *DXF* (Capítulo 2). La gran difusión de este formato garantiza que la digitalización de los mapas se pueda hacer en una amplia gama de editores gráficos que soporten este formato, nosotros hemos utilizado AutoCAD. Cada mapa contiene esencialmente poligonales y puntos distribuidos en capas que definen regiones,

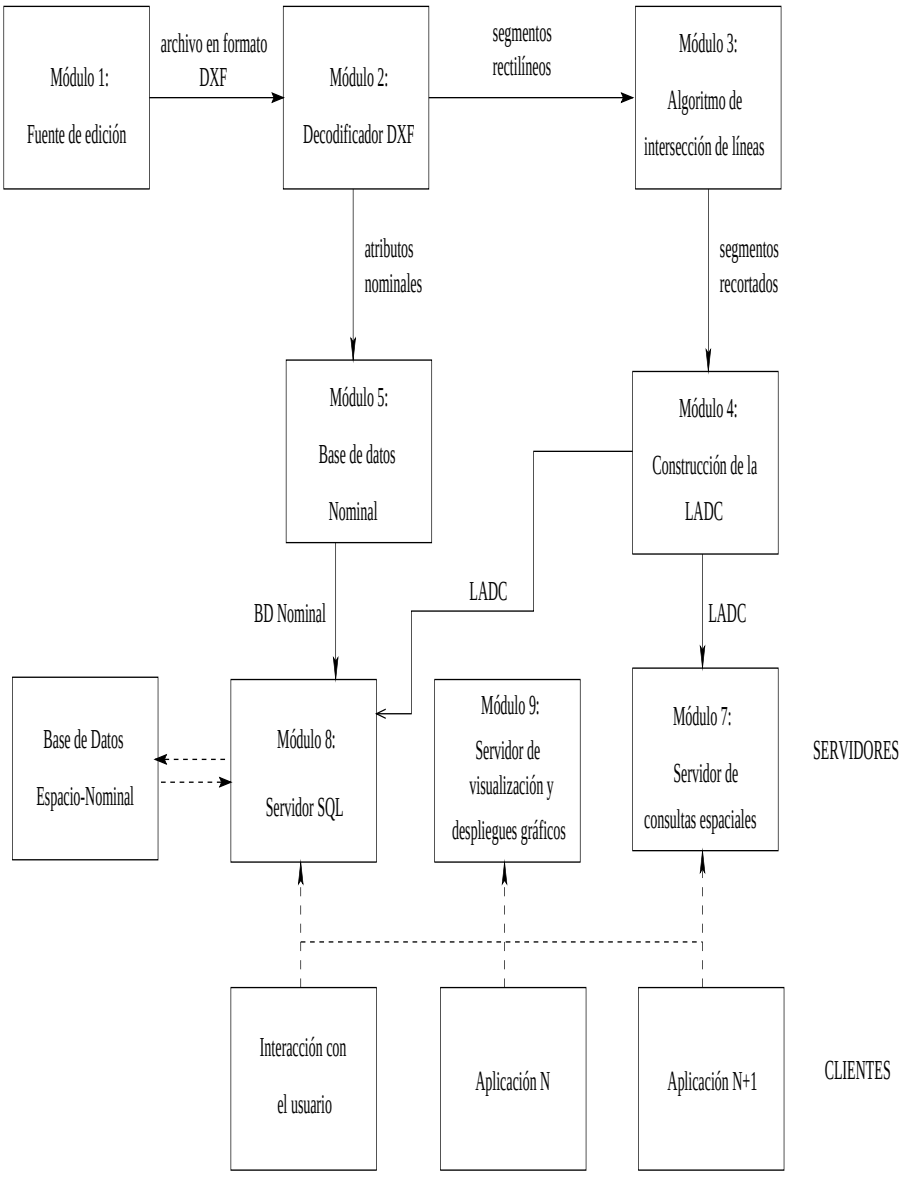


Figura 1.1: Arquitectura del núcleo de un SIG.

contornos y posiciones puntuales. A estas entidades gráficas, es posible asociarles bloques de información nominal, esto es, información referente a características no espaciales. Los mapas en formato *DXF* son decodificados por el módulo 2 del núcleo y el resultado se envía a la salida estándar que esencialmente contiene los segmentos rectilíneos que definen regiones y contornos, así como la información nominal con sus respectivas asociaciones a las entidades gráficas a las que pertenecen.
La salida del módulo de decodificación del formato *dxf* alimenta a dos procesos:

Construcción de la base de datos nominal. Es realizada por el módulo 5, y su trabajo fundamental consiste en producir las sentencias SQL para que el servidor de la base de datos genere la base nominal. Para garantizar la portabilidad, requerimos que el servidor de bases de datos soporte el estándar SQL. Los sistemas administradores de bases de datos más importantes como ORACLE, INFORMIX y SYBASE soportan este estándar; nosotros utilizamos PostgreSQL ([23]) porque se encuentra disponible como software libre bajo la licencia de GNU en Linux.

Construcción de la base de datos espacial. Este proceso consta de dos fases, la primera la lleva a cabo el módulo 3 y consiste en "limpiar" los segmentos rectilíneos preparándolos para construir la representación final (Capítulo 3); la segunda etapa es realizada por el módulo 4 y consiste en construir la estructura de representación interna que sigue un formato especial, conocido como *Lista de Aristas Doblemente Conectadas* o LADC (ver Capítulo 5 y [27]). Además de construir la LADC de cada mapa, el sistema construye la LADC del mapa formado por la superposición de todos los mapas, esto es esencial para poder hacer consultas que involucren simultáneamente varios mapas.

Los módulos de aplicación, mostrados en la parte más baja de la Figura 1.1, son procesos independientes que se comunican con el resto del núcleo mediante el modelo cliente-servidor. De hecho tenemos tres servidores:

Servidor SQL de la base de datos (Módulo 8). Las aplicaciones, como clientes de este servidor, plantean sus consultas SQL de manera directa. El cliente recibe de este módulo las tablas resultantes de la consulta.

Servidor de consultas espaciales (Módulo 7). Este servidor responde consultas relacionadas con la región que contiene a un punto dado dentro de una LADC específica. De manera más general a las regiones, contornos y puntos, contenidos en un mapa específico que caen dentro de un dominio particular. De esta manera, por ejemplo, una aplicación al detectar la selección de un punto específico sobre la ventana de interacción, puede saber a qué región dentro del mapa pertenece y a su vez utilizar este dato para consultar en el servidor SQL de la base de datos la información nominal asociada con tal región.

Servidor de visualización y despliegues gráficos (Módulo 9). Este es un programa encargado de realizar los despliegues gráficos. Tiene un lenguaje propio que permite especificar acciones como el despliegue de la LADC de un mapa específico junto con una imagen satelital en el fondo, la realización de acercamientos o alejamientos, la navegación por el mapa, el despliegue de componentes particulares del mapa utilizando atributos

específicos, entre otras. El manejo de la interacción con el usuario es atendido por un servidor adicional que depende de cada aplicación. La comunicación con este servidor se lleva a cabo con "sockets" [33] de UNIX porque en general tiene que atender las peticiones de varios clientes.

En este trabajo no pretendemos abordar en toda su extensión la implantación de este diseño pues ello excedería nuestras posibilidades de desarrollo. Fundamentalmente nos hemos concentrado en el desarrollo de los módulos 2,3, 4 y 7. El resto de este capítulo será dedicado a especificar con mayor detalle las características funcionales de estos módulos.

1.2 Algoritmo de intersección de segmentos de línea.

Este módulo se encarga de preparar los segmentos resultantes de la decodificación del formato *DXF* para construir la representación interna de los mapas. El objetivo es detectar las intersecciones entre segmentos no expresadas explícitamente y modificar el mapa original para que no se den este tipo de situaciones.

Un par de poligonales en un mapa pudieran intersectarse sin que tal intersección se encuentre registrada como punto extremo de algún segmento en el mapa. Por ejemplo, el segmento que va del punto $(0,0)$ al $(1,1)$ y el segmento que va del punto $(1,0)$ al punto $(0,1)$ se intersectan en el punto $(0.5,0.5)$; sin embargo, este punto no es un extremo de alguno de los dos segmentos originales. Las estructuras de búsqueda utilizan generalmente los puntos de intersección entre segmentos como referencia para decidir su estrategia de localización; su funcionamiento se simplifica si las intersecciones entre segmentos coinciden con sus puntos extremos. En general, esto se puede lograr si subdividimos adecuadamente los segmentos del mapa original. En el ejemplo citado el mapa resultante de la subdivisión contendría los segmentos $(0,0)$ a $(0.5,0.5)$, $(0.5,0.5)$ a $(1,1)$, $(0,1)$ a $(0.5,0.5)$ y $(0.5,0.5)$ a $(1,0)$.

El proceso de subdivisión es muy importante para definir con exactitud las regiones de un mapa y así efectuar consultas o búsquedas sobre ellas. Por ejemplo, supongamos que se necesita construir una aplicación donde después de seleccionar con el cursor del ratón una región, queremos que aparezcan datos asociados a la misma como son su nombre, número de habitantes, altura sobre el nivel del mar, etc. En primer lugar necesitamos saber cuál es esa región, lo que se hace consultando al servidor de consultas espaciales (módulo 7) y en segundo lugar necesitamos la llave de esa región para extraer la información nominal asociada con la ayuda del servidor SQL (módulo 8). El servidor de consultas espaciales sería incapaz de determinar la región si no se han encontrado TODAS las intersecciones (explícitas y no explícitas).

Una de las operaciones fundamentales de un SIG es la superposición de mapas para poder responder consultas elaboradas. Por ejemplo, si se dispone de un mapa topográfico con las curvas de nivel a cada 100 metros y un mapa de las temperaturas promedio anuales de cierta región de estudio, podemos preguntar sobre las regiones que estén entre los 800 y 1,000 metros sobre el nivel del mar y cuya temperatura promedio anual oscile entre los 20^oC y 25^oC. El algoritmo de intersección de segmentos de línea es muy importante para llevar a cabo esta operación, ya que construye un nuevo mapa que es el resultado de la superposición

de los mapas originales con la característica fundamental de que cualquier segmento de línea sólo tiene intersecciones a lo más en sus extremos.

Formalmente, la superposición S del mapa A con el mapa B, debe cumplir lo siguiente (la generalización a más de dos mapas es inmediata):

1. Cada región del mapa S está contenida en una sola región del mapa A y una sola región del mapa B.

2. No existe otro mapa S' que satisfaga la condición anterior y que además contenga una región r' que a su vez contenga propiamente una región r de S.

En el capítulo 3 y 4, respectivamente, tratamos con más detalle dos algoritmos de intersección de segmentos de línea: el método de Bentley/Ottmann y una extensión al método de Edelsbrunner.

1.3 Construcción de la LADC (lista de aristas doblemente conectadas)

Una gráfica $G = (V, E)$ [32] es un par ordenado de conjuntos disjuntos V, E tal que $E \subseteq V^2$ y $V \neq \emptyset$. El conjunto V es el conjunto de *vértices* de G y E es el conjunto de *aristas*. Una arista une a los vértices μ y ν y es denotado por (μ, ν).

Un *encaje planar* de $G = (V, E)$ (ver [7] y Capítulo 5) es una transformación ψ que asocia cada vértice v en V con un punto $\psi(v)$ en el plano y cada arista $e = (\mu, \nu)$ con una curva simple que tiene como extremos a $\psi(\mu)$ y $\psi(\nu)$ de modo tal que la transformación de dos aristas diferentes sólo puede intersectarse en sus puntos extremos. Una gráfica es planar si admite un encaje planar.

Una LADC (Lista de Aristas Doblemente Conectadas) es una estructura para representar una gráfica planar.

Sea $V = \{v_1, ..., v_N\}$ y $E = \{e_1, ..., e_M\}$. La componente principal en la LADC de una gráfica planar $G = (V, E)$ es el *nodo lado* que consiste en un arreglo con $\mid E \mid$ entradas, hay una correspondencia uno a uno entre las aristas y las entradas de este arreglo. Una entrada consiste de cuatro campos de información V_1, V_2, C_1 y C_2, y dos campos apuntador P_1 y P_2. El significado de los campos es el siguiente. El campo V_1 contiene el origen de la arista y V_2 el fin; de esta manera la arista recibe una orientación convencional. Los campos C_1 y C_2 contienen los nombres de las caras ubicadas, respectivamente, en el lado izquierdo y derecho de la arista orientada de V_1 a V_2. El apuntador P_1 (respectivamente P_2) apunta al primer nodo lado encontrado después de que la arista (V_1, V_2) es girada en sentido contrario al de las manecillas del reloj alrededor de V_1 (respectivamente V_2). Los valores de las caras, vértices, aristas y orientación de las mismas son tomados aleatoriamente. En la Figura 1.2 se muestra el fragmento de una gráfica y en la Tabla 1.1 su LADC correspondiente.

En el capítulo 5 definimos con más detalle el concepto de encaje planar, su representación mediante una LADC y el método de construcción a partir de la salida del módulo 2.

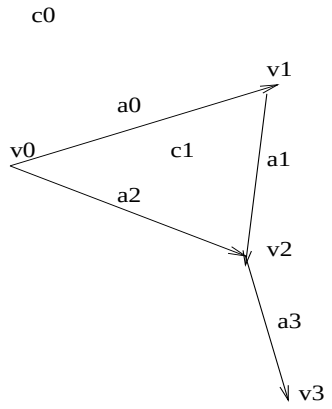


Figura 1.2: Un encaje planar (con aristas dirigidas de manera arbitraria).

arista	V_1	V_2	C_1	C_2	P_1	P_2
a0	0	1	0	1	2	1
a1	1	2	0	1	0	2
a2	0	2	1	0	0	3
a3	2	3	0	0	1	3

Tabla 1.1: LADC correspondiente a la Figura 1.2.

1.4 Asociación de atributos nominales a un mapa

Un mapa puede tener una de tres orientaciones posibles (ver [8], [12] y [11]): a regiones, a líneas y a puntos. Así en un mapa orientado a regiones, por ejemplo, cada región tiene asociados atributos que no dependen de su ubicación espacial, a estos atributos se les llama *nominales*. De esta manera, podemos asociar información tal como la altura sobre el nivel del mar en un mapa topográfico, de temperaturas en un mapa de isotermas o la información estadística en un mapa de población. Los atributos nominales deben formar parte de la Base de Datos espacio-nominal para que posteriormente podamos realizar consultas al sistema.

Hay varias formas de asociar atributos nominales a un mapa, por ejemplo, el Sistema ArcView [19] ofrece un sistema de asociación dinámico que permite crear la estructura de las relaciones y asociarlas directamente a los elementos espaciales. En nuestro caso, utilizamos directamente bloques de AutoCAD, donde se almacenan los datos nominales que se asocian con los elementos espaciales.

Sumario

En el capítulo se propuso una arquitectura particular de lo que hemos denominado *núcleo de un Sistema de Información Geográfica* o simplemente *núcleo*. Los módulos de la arquitectura propuesta permiten llevar a cabo la representación, manipulación y consulta de la información espacio-nominal de un mapa, explicando en mayor detalle aquellos módulos que

tienen que ver con el tratamiento de la información espacial, ya que éste es el objetivo de desarrollo en la tesis.

Cabe señalar que los módulos de la arquitectura propuesta se comunican entre sí utilizando el modelo de comunicación cliente-servidor, un modelo que permiten de manera natural sistemas operativos como UNIX.

Capítulo 2

Extracción de información espacio-nominal de un mapa en formato *DXF*

El objetivo de este capítulo es mostrar cómo se representa en el formato *DXF* la información espacial y nominal de un mapa. También se muestra el autómata que decodifica mapas en este formato y extrae la información espacio-nominal que contenga.

El resultado final ha sido la producción de un programa que decodifica la información tanto espacial como nominal de un mapa en formato *DXF*. Este programa toma de su entrada estándar un archivo *DXF*; en los parámetros de la línea de comandos se especifican los filtros necesarios para extraer la información requerida. El código fuente de este programa forma el módulo 2 del núcleo de un SIG, tal como ha sido representado en el Capítulo 1.

2.1 Almacenamiento de información espacio-nominal dentro de un archivo en formato *DXF*

El estándar *DXF* ("Drawing Interchange File"), fue introducido por AutoCAD como un formato para intercambiar información entre sistemas de dibujo y diseño asistidos por computadora [4], aquí nos hemos basado en la especificación para la versión 14 de *DXF*. Este formato es en verdad extenso y aun se vislumbran nuevas extensiones, a nosotros tan solo nos interesa su capacidad para representar información espacial y nominal. Ahora daremos una breve introducción a la terminología de *DXF* que usamos en este trabajo.

En el formato *DXF* la información espacial se representa mediante *entidades gráficas* que a su vez contienen *atributos*. Por ejemplo, un círculo, un segmento de línea o una poligonal, son entidades gráficas cuyos atributos contienen: las coordenadas del centro y el radio para el círculo, las coordenadas de los extremos del segmento de línea, o las coordenadas de los puntos que definen la poligonal.

2.2 Representación de información nominal

La forma más simple de representar información nominal (es decir, información no espacial y capaz de ser representada por un Sistema Manejador de Bases de Datos común) en *DXF* es mediante bloques. Un *bloque* es una estructura de datos en el sentido clásico del término (ver [24]), es decir, un bloque es una familia de *variables* o *atributos* que se identifican bajo un mismo nombre y de la cual se pueden crear tantas instancias como sea necesario. Los atributos de un bloque pueden ser de dos tipos: espaciales y nominales. Un atributo espacial es aquel que corresponde con una primitiva gráfica y uno nominal corresponde a un valor con un dominio específico y de manera genérica se representan mediante cadenas de caracteres. A cada *instancia de un bloque* se le denomina *inserción* y cada inserción tiene una posición espacial (X, Y, Z) llamada *punto de inserción*. Estas coordenadas siempre forman parte de los atributos de cualquier bloque.

En general, la información en *DXF* se distribuye en *capas* especificadas por el usuario, de esta manera es posible estructurar los componentes de un dibujo.

Así, desde el punto de vista de este capítulo, un *mapa* es un conjunto de entidades gráficas, un conjunto de bloques, y un conjunto de instancias de esos bloques, que se encuentran almacenados en un grupo de capas de un archivo *DXF*, cuyo propósito es el de representar a un encaje planar (ver Capítulo 5) junto con información nominal asociada a las caras, aristas y vértices del mismo.

Cada mapa puede tener uno de tres tipos fundamentales: *regiones*, *líneas* y *puntos*. En un mapa de regiones la información nominal se asocia con las caras del encaje planar, en un mapa de líneas la información nominal se asocia con las aristas del encaje planar y en un mapa de puntos la información nominal se asocia con los vértices del encaje planar. Por ejemplo, los mapas de división política, vegetación o geología son mapas de regiones; los mapas de curvas de nivel e isotermas son mapas de líneas y los mapas que registran la ubicación geográfica de poblaciones en una escala muy grande son mapas de puntos.

Los bloques no fueron creados específicamente para representar información nominal, de hecho no tienen la riqueza de un Lenguaje de Definición de Datos (ver [21]). Sin embargo, usando bloques es posible asociar fácilmente información nominal a mapas existentes. De esta manera nos hemos ahorrado tener que desarrollar un editor dedicado exclusivamente a hacer esta asociación.

2.3 Manejo de bloques en AutoCAD

No pretendemos explicar aquí cómo utilizar AutoCAD para producir mapas (esto se explica ampliamente en textos como [24, 10]). Sin embargo, debido a que el uso de bloques no es frecuente, y nuestro modelo de asignación de información nominal recae totalmente en esta facilidad del estándar *DXF*, creemos conveniente introducir los mecanismos básicos para la definición e inserción de bloques en un dibujo.

A continuación damos los comandos básicos para la creación e inserción de bloques, así como la definición y redefinición de sus atributos.

`ATTDEF`. Comando para definir atributos. Los atributos son una de las entidades básicas

en los dibujos de AutoCAD que contienen información textual que a su vez puede ser guardada como parte la instancia de un bloque. Un atributo es equivalente a un campo particular en una tabla del modelo relacional. Al insertar un bloque con atributos, AutoCAD le preguntará al usuario por cada uno de los valores de estos atributos. Dependiendo del valor de la variable `ATTDIA` [1] el usuario podrá especificar los atributos desde la línea de comandos o desde una ventana propia para la edición de los mismos.

`_DDATTE`. Comando para redefinir atributos.

`BLOCK`. Este comando permite combinar varias entidades de un dibujo en un solo grupo que, bajo un mismo nombre, puede manipularse como un todo. Para crear un bloque es necesario crear primero las entidades gráficas y atributos que debe contener. Una vez creado un bloque, AutoCAD elimina del espacio de dibujo los elementos que lo conforman y cada vez que se inserte se crea una copia de los elementos que contiene vistos como una unidad.

`INSERT` Este comando inserta un bloque previamente definido en el dibujo. Al hacer una inserción AutoCAD solicita al usuario que proporcione el punto de inserción en el espacio de dibujo así como los valores de los atributos que contiene. Cada inserción del bloque puede tener diferentes factores de escala y ángulos de rotación. Para modificar un bloque primero se inserta en algún lugar, luego se separan sus partes con el comando `EXPLODE`, se modifica y se define nuevamente el bloque. Todas las inserciones pasadas y futuras registrarán la modificación. Para modificar los atributos del bloque se usa el comando `DDEDIT`.

2.4 Edición de mapas y asociación de información nominal

Para que un mapa pueda ser utilizado con nuestras herramientas, es necesario que cumpla ciertas condiciones de calidad que aseguren la no existencia de ambigüedades en la información.

En cuanto a la edición de primitivas gráficas se requieren las siguientes condiciones:

`P1:` Cada archivo *DXF* puede contener un número arbitrario de mapas, pero cada mapa debe estar contenido en exactamente un archivo *DXF*.

`P2:` Las primitivas gráficas que se aceptan en los mapas son: líneas, polilíneas y bloques. Cualquier otra primitiva es ignorada.

`P3:` En los mapas de regiones cada poligonal deberá estar **perfectamente cerrada** y deberán evitarse los cierres en falso y el efecto del "pulso tembloroso". En cada mapa de líneas las secuencias de segmentos asociados a una misma entidad geográfica deberán formar parte de la misma polilínea.

[1] El valor de la variable `ATTDIA` puede cambiarse con `SET ATTDIA <valor>` donde `<valor>` es 1 o 0

`P4:` Cada mapa deberá contener un solo tipo de bloque debido a que, como se menciono en el Capítulo 1, los mapas sólo pueden tener una de tres orientaciones posibles: a regiones, a líneas o a puntos.

`P5:` El usuario debe tener en cuenta que los nombres de bloque que utilice serán a su vez utilizados como nombres de esquemas de relación en el Sistema Manejador de Bases de Datos que se esté utilizando, así que debe evitar que se dupliquen estos nombres dentro de una misma aplicación.

`P6:` Las inserciones deberán hacerse de manera precisa sobre los elementos del mapa de acuerdo al tipo que el mapa tenga. Así en un mapa de regiones los puntos de inserción estarán dentro de las regiones del mapa, en un mapa de líneas los puntos de inserción estarán sobre alguno de los vértices que definen las poligonales del mapa y en un mapa de puntos sobre los vértices. En un mapa de regiones cada región deberá contener una y solo una inserción, condiciones similares deberán cumplirse para los otro tipos de mapas.

`P7:` Entre los atributos de un bloque utilizado para introducir datos nominales siempre existirá un atributo invisible y constante llamado *definición* que tendrá el valor siguiente:

`<TIPO_DE_MAPA>, <DEFINICION_ATRIBUTO`$_1$`>,...,<DEFINICION_ATRIBUTO`$_N$`>`

Donde `<TIPO_DE_MAPA>` es alguno de los valores `<REGION_DCEL>`, `<LINEA_DCEL>` o `<PUNTO_DCEL>`, dependiendo de si el mapa es de regiones, de líneas o de puntos, respectivamente. `<DEFINICION_ATRIBUTO`$_i$`>` con ($1 \leq$ `i` $\leq$ `N`) es la definición del atributo i-ésimo del bloque en el formato que use el comando ``‘‘CREATE TABLE’’`` del servidor SQL que se esté utilizando [3] para definir los atributos de un esquema de relación.

Como ejemplo de la condición `P6`, supongamos que se le asignara el valor ``‘‘REGION_DCEL, did DECIMAL(3), name VARCHAR(40) CONSTRAINT con1 CHECK (did > 100 AND name > ’’)`` al campo *definición* de un bloque llamado “estado”. Entonces el programa *dxf* (que como se explicó al principio de este Capítulo extrae la información espacio-nominal de un mapa en un archivo *DXF*) produce en la salida la siguiente secuencia:

```
CREATE TABLE estado (REGION_DCEL decimal(10),
                     X           float,
                     Y           float,
                     Z           float,
                     did         decimal(3),
                     name        varchar(40) CONSTRAINT con1
                                 CHECK (did > 100 AND name > '')
                    )
```

EL atributo ``‘‘REGION_DCEL’’`` de esta tabla sirve para guardar el número de la región (cara) que contiene al punto de inserción del bloque dentro de la LADC del encaje (ver Sección 6.1). Las coordenadas mismas del punto de inserción se almacenan en los atributos X, Y y Z de la tabla. Los demás atributos de la tabla provienen del valor asignado al atributo *definición* del bloque *estado*.

CÓDIGO DE GRUPO	VALOR DE GRUPO
0	Identifica el inicio de una entidad
2	Identifica una sección
8	Nombre de la capa
10	Primera coordenada X (de una línea, polilínea, circulo, etc.)
11-18	Otras coordenadas X
20	Primera coordenada Y
21-28	Otras coordenadas Y
30	Primera coordenada Z
31-38	Otras coordenadas Z
40-48	Valores en punto flotante (altura del texto, factor de escala, etc.)
50-58	Ángulos
62	Color
999	Comentarios

Tabla 2.1: Algunos códigos y valores de grupo de un archivo *DXF*.

El programa *DXF* no realiza ningún tipo de revisión sintáctica para determinar si la sintaxis del campo *definición* de un bloque es correcta, quien finalmente detecta los errores es el servidor SQL.

2.4.1 Estructura general de un archivo *DXF*

En esta sección se describe la estructura general de un archivo *dxf* con el propósito de mostrar la información técnica requerida para decodificarlo. Un archivo *DXF* está compuesto por *grupos* y dividido en *secciones*.

Códigos y valores de grupo de un archivo *DXF*

En un archivo *DXF* un *grupo* es una pareja de líneas consecutivas (impar, par). La primer línea de un grupo contiene un entero positivo distinto de cero llamado *código de grupo*. La segunda línea es el *valor de grupo* y su formato depende del código de grupo especificado. En la Tabla 2.1 se muestran algunos códigos y valores de grupo. Así, el código de grupo 999 indica que la línea siguiente es un comentario; los códigos de grupo 10, 20 y 30 indican, respectivamente, que siguen los valores X_1, Y_1 y Z_1 de una entidad dada (por ejemplo un segmento de línea).

Secciones de un archivo *DXF*

Un archivo *DXF* está compuesto de por lo menos cuatro secciones conformadas por grupos. A continuación se muestra de manera resumida el contenido de un archivo *DXF*.

0 *(Inicio de la Sección HEADER)*
SECTION
2
HEADER
Las variables Header van aquí
0
ENDSEC (Fin de la Sección HEADER)

0 *(Inicio de la Sección TABLES)*
SECTION
2
TABLES
0
TABLE
2
VPORT
70
Las tablas de viewport van aquí
0
ENDTAB
0
TABLE
2
APPID, DIMSTYLE, LTYPE, LAYER, STYLE, UCS, VIEW, oVPORT
70
Tablas van aquí
0
ENDTAB
0
ENDSEC
2
BLOCKS
Definiciones de entidades de bloque
0
ENDSEC Fin de sección bloques
0 *Inicio de sección entidades*
SECTION
2
ENTITIES
Las entidades gráficas van aquí
0
ENDSEC Fin de sección entidades
0
EOF Fin de archivo

CÓDIGO DE GRUPO	SIGNIFICADO
6	Nombre del tipo de línea
38	Elevación
62	Número de color

Tabla 2.2: Algunos códigos asociados a entidades..

Las secciones que componen un archivo *DXF* son las siguientes:

1. Sección `HEADER`. En esta sección se encuentra la información general del dibujo que incluye datos tales como como el número de versión de AutoCAD o las variables de sistema.

2. Sección `TABLES`. Esta sección contiene la definición de las siguientes tablas:

   (a) `LTYPE` (Tipo de línea)

   (b) `LAYER` (Layer)

   (c) `STYLE` (Estilo de línea)

   (d) `VIEW` (Vista)

   (e) `UCS` (Sistema de coordenadas de usuario)

   (f) `DIMSTYLE` (Dimensión de estilo)

   (g) `APPID` (Identificación de aplicación).

   El orden de las tablas puede cambiar, pero la tabla `LTYPE` siempre precede a la tabla `LAYER`. Cada tabla es introducida con el grupo 0 con la etiqueta `LABEL`. Esto es seguido por el grupo 2 que identifica a una tabla en particular y el grupo 70 que especifica el número máximo de entradas en la tabla que pueden seguir.

3. Sección `BLOCKS`. Esta sección contiene definiciones de los bloques asociados a las primitivas gráficas en AutoCAD. Nos interesa decodificar esta sección porque contiene los bloques y las inserciones de información nominal.

4. Sección `ENTITIES`. Aquí se almacenan las primitivas o entidades gráficas del archivo, tales como líneas, círculos, arcos, etc. Nos interesa decodificar esta sección porque contiene la información de los segmentos de línea que componen el mapa. Cada entidad inicia con el grupo 0 que sirve para identificarla, la capa a la que pertenece tiene código de grupo 8; asimismo, las entidades pueden tener elevación, tipo de línea o color asociados con ellas. Las entidades en general tienen un identificador numérico único dentro del archivo *DXF* al que llamamos "asa" (handle) y lo utilizamos para establecer vínculos entre la información nominal con la espacial; el código de grupo para el asa es 5. En la Tabla 2.2 se dan algunos códigos asociados a entidades.

   A continuación se muestra cómo se describen algunas entidades.

```
LINE    10, 20, 30, X1, Y1, Z1, 11, 21, 31, X2, Y2, Z2
POINT   10, 20, 30, X1, Y1, Z1
CIRCLE  10, 20, 30, X1, Y1, Z1, 40 R
ARC     10, 20, 30, X1, Y1, Z1, 40 R, 50 ángulo inicial, 51 ángulo final
```

Donde `X1, Y1, Z1, X2, Y2, Z2` y `R` son valores reales positivos.

De las secciones de una archivo *DXF* nos interesan particularmente `ENTITIES` y `BLOCKS` ya que éstas contienen la información espacio-nominal del mapa.

2.4.2 Autómata para la extracción de información espacio-nominal de un mapa

En la Figura 2.1 se muestra el decodificador del formato *DXF*. El Lenguaje *DXF* es regular y por consiguiente su análisis sintáctico puede ser llevado a cabo por un autómata finito determinístico (ver [18]). En la Figura 2.1 presentamos una versión esquemática global de este autómata, el código completo del mismo puede se consultado en el programa en lenguaje "C" *dxf.c* del paquete de distribución de los programas (ver Apéndice A). En la Figura 2.1 se ha utilizado la simbología siguiente: Los círculos representan sub-autómatas con tareas de decodificación específicas indicadas por las etiquetas contenidas dentro de los círculos. Las transiciones entre sub-autómatas se representan mediante aristas etiquetadas con los códigos de operación del formato *DXF*. Los hexágonos mostrados dentro del rectángulo que aparece en esta figura, representan el conjunto de códigos atómicos *DXF*, junto con sus valores asociados, que deben analizar los sub-autómatas para cumplir sus tareas de decodificación. El estado inicial del autómata está etiquetado como "INICIO" y el final se ha representado con doble circulo concéntrico.

Estrictamente hablando, este autómata funciona como una máquina de Moore (ver Capítulo 2 de [18]). Para reducir el tamaño del autómata se ha incluido tan sólo un estado por código de operación, así como un conjunto de variables internas que registran el sub-autómata que actualmente realiza la decodificación.

El programa *DXF* permite llevar a cabo la extracción de información espacial y nominal de un mapa en formato *DXF*. Hemos usado este decodificador para alimentar los algoritmos desarrollados en la tesis, sin embargo es una herramienta de propósito general que fácilmente puede ser extendida para atender otro tipo de aplicaciones que requieran extraer información de archivos en formato *DXF*.

Sumario

En este capítulo se mostró como es representada la información espacio-nominal en un mapa en formato *DXF* y se mostró el autómata que permite extraer dicha información para posteriormente ser procesada por los módulos respectivos. La implementación del autómata puede verse en el programa en lenguaje C contenido en el disco compacto anexo al documento de tesis. Cabe mencionar que cuando se llevo a cabo este desarrollo, aún no estaba presente en

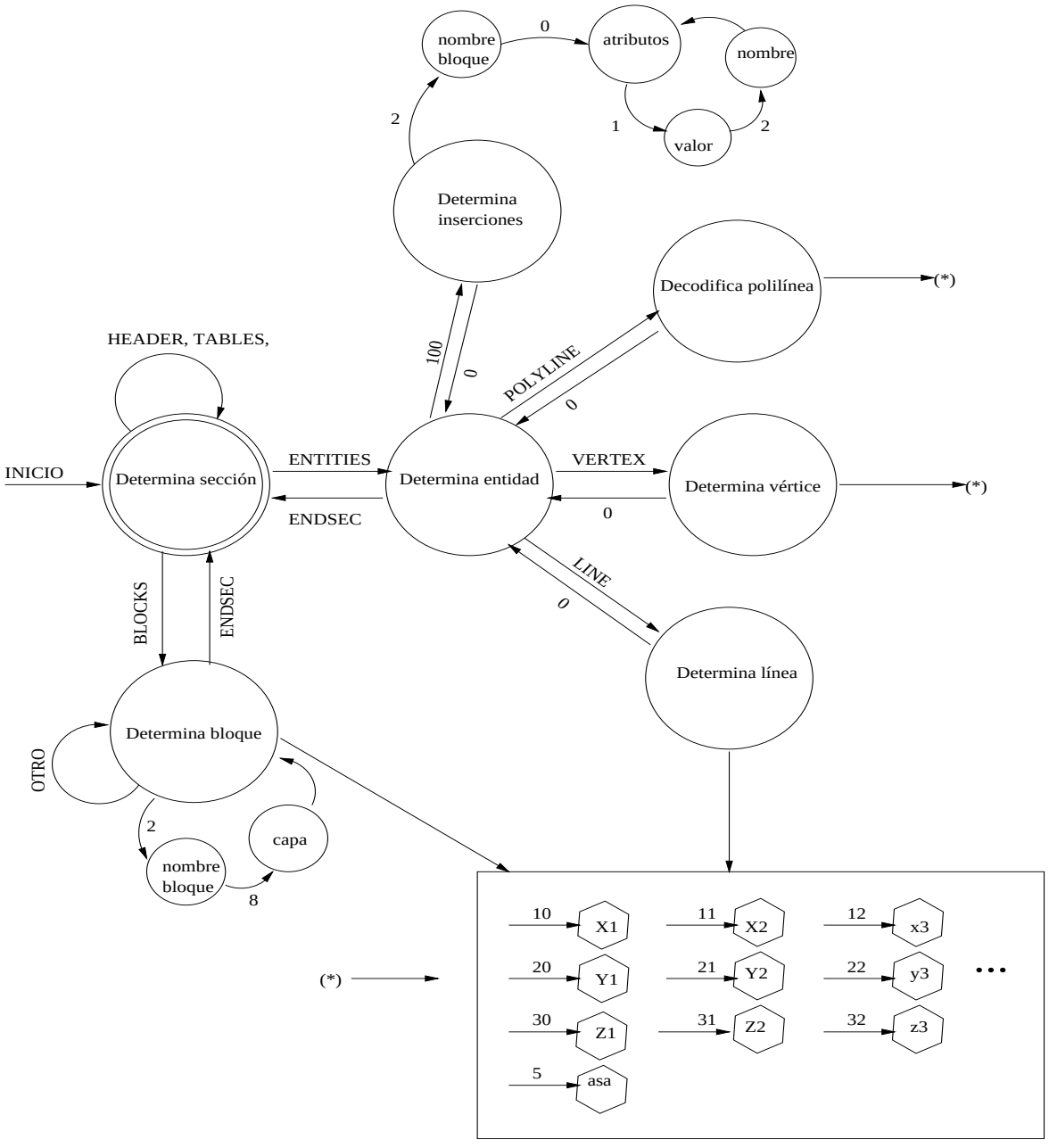


Figura 2.1: Decodificador del formato *DXF*.

el mercado AutoCAD-Map que permite, entre otras cosas, llevar a cabo una asoción distinta de los bloques de información nominal.

Capítulo 3

Algoritmo de intersección de segmentos de línea

3.1 Introducción

Dados n segmentos de línea en el plano, nuestro objetivo es determinar todas la intersecciones entre pares de segmentos. Esto es importante porque, como explicamos en el Capítulo 5, al construir la representación espacial de un mapa, dos segmentos cualesquiera se deben intersectar a lo más en sus extremos. Este problema puede ser resuelto mediante el método directo de probar todas las parejas posibles de segmentos; este proceso tiene una complejidad $O(n^2)$ y, si consideramos que típicamente un mapa contiene varias decenas de miles de segmentos, será fácil entender que el tiempo requerido puede ser extremadamente grande. En la literatura se ha atacado el problema de reducir la complejidad del método directo [16] y lo mejor que se conoce es el método de Bentley/Ottmann [5]. Este método, aunque correcto es impráctico, y ha tenido que ser modificado. Al método modificado le hemos llamado *extensión al método de Edelsbrunner*.

El método de Bentley/Ottmann tiene complejidad temporal de orden $O(n \lg n + k \lg n)$, donde k es el número de intersecciones encontradas. A pesar de ser un algoritmo óptimo, este método involucra decisiones que dependen de cálculos aritméticos de punto flotante, así el método resulta muy sensible a los errores de redondeo. Al instrumentarlo y probarlo con grandes mapas, descubrimos errores que no se pueden resolver de una manera general. En la Sección 3.2.3 damos ejemplos concretos de las dificultades que presenta.

La extensión al método de Edelsbrunner reporta las intersecciones de un conjunto dado de segmentos rectilíneos sobre el plano utilizando el método de Edelsbrunner, que reporta las intersecciones de un conjunto dado de rectángulos sobre el plano (Capítulo 4 y [28]). La extensión al método de Edelsbrunner es un método con la misma eficiencia del algoritmo original de Bentley/Ottmann, pero sin sus problemas de punto flotante.

La extensión al método de Edelsbrunner logra resolver el problema de la aritmética de punto flotante del algoritmo de Bentley/Ottmann, además de que nos ofrece un algoritmo eficiente cuya complejidad temporal es de $\theta(NlogN+s)$, donde s es el número de rectángulos asociados a segmentos que se intersectan, más no sus segmentos respectivos. En el Capítulo 4 se presenta de manera más detallada la extensión al método de Edelsbrunner.

3.2 Método de Bentley/Ottmann

3.2.1 Descripción del algoritmo

Sea S un conjunto de segmentos en el plano que no contiene segmentos verticales y x un número real. Para cada $s \in S$ denotamos por $i(s)$ al punto extremo en s con menor abscisa y a $f(s)$ al punto extremo con mayor abscisa.

Decimos que un segmento de línea en el plano s es *comparable en la abscisa* x si la línea vertical que pasa por la abscisa x intersecta a s en un punto diferente a $f(s)$. El segmento $s1$ está *arriba de* $s2$ (o $s2$ está *abajo de* $s1$) con respecto a x, escrito $s1 >_x s2$, si $s1$ y $s2$ son comparables en la abscisa x y el punto de intersección de $s1$ con la línea vertical l que pasa por la abscisa x tiene ordenada mayor que el punto de intersección de $s2$ con l. En la Figura 3.1 la relación de orden entre los segmentos de línea $s1$, $s2$, $s3$ y $s4$ es la siguiente:

$$s2 >_u s4,\ s1 >_v s2,\ s2 >_v s4 \text{ y } s1 >_v s4.$$

El segmento $s3$ no es comparable con ningún otro.

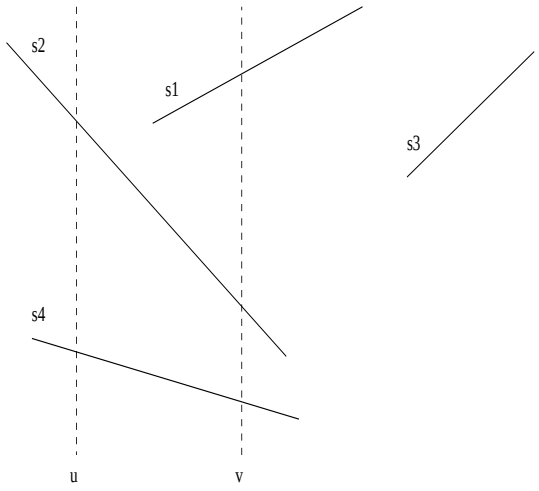


Figura 3.1: Relación de orden entre segmentos de línea.

Al conjunto de segmentos en S comparables en la abscisa x se les denota por $\sigma(x)$. Para cada x la relación $>_x$ es un orden total en $\sigma(x)$. Observamos que si x es menor que la máxima abscisa de los segmentos en S entonces existe un número real $\epsilon_x > 0$ mínimo para el cual $\sigma(x) \neq \sigma(x + \epsilon_x)$ o la relación $>_x \neq >_{x+\epsilon_x}$ entonces se presenta al menos una de las siguientes condiciones:

C1: Existe al menos un segmento $s \in \sigma(x + \epsilon_x) - \sigma(x)$ (se alcanza al menos un nuevo extremo izquierdo).

C2: Existe al menos un segmento $s \in \sigma(x) - \sigma(x + \epsilon_x)$ (se alcanza al menos un extremo derecho).

C3: Se cumple que $\sigma(x) = \sigma(x + \epsilon_x)$ pero $>_x \neq >_{x+\epsilon}$. Este caso se presenta cuando en la abscisa $x + \epsilon_x$ se intersectan dos segmentos.

El algoritmo original de Bentley/Ottmann inicia tomando la abscisa mínima x_0 de los segmentos en S. A continuación construye los conjuntos $\sigma(x_0), \sigma(x_1), \ldots, \sigma(x_n)$ donde $x_{i+1} = x_i + \epsilon_{x_i}$ para $i = 0, \ldots n-1$, donde n es tal que x_n corresponde con la máxima abscisa de los segmentos en S.

Desde un punto de vista algorítmico, el conjunto $\sigma(x_0)$ se representa en una estructura llamada "línea de barrido", para $0 \leq i \leq n-1$ se construye, a partir de la representación en la línea de barrido de $\sigma(x_i)$, la representación de $\sigma(x_i)$ de una manera dinámica. La línea de barrido es simplemente un diccionario de búsqueda que permite insertar y borrar segmentos, así como localizar las modificaciones de la relación $<_x$ en tiempo logarítmico. Dada la abscisa x_i $(1 \leq i \leq n)$ si aparece una condición **C1**, se inserta el segmento correspondiente en el diccionario respetando el orden $<_{x_i}$, si se presenta la condición **C2** entonces se borra el segmento asociado y finalmente si ocurre la condición **C3** entonces se modifica la relación de orden y se reporta la intersección. Hay que tener en cuenta que también en las condiciones **C1** y **C2** pueden presentarse intersecciones entre segmentos (en sus extremos) así que también estas intersecciones deben ser reportadas. El algoritmo final de Bentley/Ottmann se ha escrito de modo que sea posible incorporar segmentos verticales.

En la Figura 3.2 se muestra (con línea punteada) el proceso que sigue el algoritmo de Bentley/Ottmann deteniéndose en las abscisa x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 y x_9. Podemos ver que en x_1 se alcanza la condición **C1** (ya que $\sigma(x_1) = \{s1\}$, $\sigma(x_2) = \{s1, s2\}$ y $\epsilon_x = x_2 - x_1$), en x_7 se alcanza la condición **C2** (ya que $\sigma(x_7) = \{s1, s3\}$, $\sigma(x_6) = \{s1, s2, s3\}$ y $\epsilon_x = x_7 - x_6$) y en x_6 se alcanza la condición **C3** (ya que $\sigma(x_5) = \{s1, s2, s3\}$, $\sigma(x_6) = \{s1, s2, s3\}$ y $\epsilon_x = x_6 - x_5$, pero $>_{x_5} \neq >_{x_6}$ $(s1 >_{x_5} s2,\ s2 >_{x_6} s1)$).

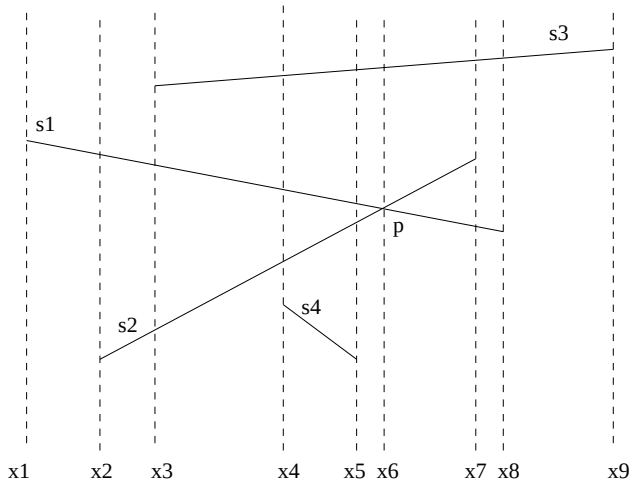


Figura 3.2: Proceso de la línea de barrido.

La construcción de los conjuntos $\sigma(x_i)$ se realiza de manera secuencial, partiendo desde $i = 0$ hasta $i = n$, proceso denominado *movimiento* de la línea de barrido; la *abscisa actual de la línea de barrido* es la abscisa x_i del conjunto $\sigma(x_i)$ que la línea de barrido representa

en el momento actual. El objetivo fundamental del movimiento de la línea de barrido es identificar, en cada paso, cuando ocurre la condición **C3**. Es decir, reportar las intersecciones entre segmentos. La naturaleza secuencial del método, así como el hecho de que al arribar a un conjunto $\sigma(x_i)$ puedan ocurrir más de una de las condiciones **C1**, **C2** o **C3**, obliga a que el movimiento de la línea de barrido se tenga que hacer con mucho cuidado para evitar perder puntos de intersección. Los detalles para garantizar que el proceso se realiza correctamente se describen a continuación.

Algorithm 3.2.1 *Algoritmo de intersección de segmentos de línea en el plano.*

Entrada: *N segmentos de línea en el plano.*

Salida: *Todos los pares de intersecciones de los N segmentos de línea en el plano.*

Método:
1.$\mathcal{E} \leftarrow$ *(El conjunto de extremos de segmentos en S, ordenados ascendentemente en x)*
2.$\mathcal{L} \leftarrow \emptyset$ *(*$\mathcal{L}$ *representa a la línea de barrido)*
*3.***for** *cada punto p en* $\mathcal{E}$ *(siguiendo el orden ascendente)* **do**
4. **if** *p es extremo izquierdo del segmento s* **then** *(Condición* **C1***)*
5. *insertar s en* $\mathcal{L}$*;*
6. *verificar si s intersecta los segmentos inmediatamente arriba y abajo de él en* $\mathcal{L}$*, y de ser así, (digamos que intersecta a t), agregar el punto de intersección de s y t a* $\mathcal{E}$ *(ordenado con respecto a x);*
8. **else if** *p es el extremo derecho de un segmento s* **then**
9. **if** *el punto de intersección del par de segmentos directamente arriba y abajo de s no esta en* $\mathcal{E}$ **then**
10. verificar si existe intersección y, de ser así, agregar el punto de intersección a Q (respetando el orden en x)
11. borrar s de $\mathcal{L}$*; (Condición* **C2***)*
12. **else** *(p es la intersección de s y t) (Condición* **C3***)*
13. reportar el par como intersección;
14. intercambiar las posiciones de s y t en $\mathcal{L}$*;*
15. verificar intersección del segmento superior (s ó t) con el segmento de
16. arriba e intersección del segmento inferior (s ó t) con el de abajo;

Nótese que una condición necesaria para que dos segmentos $s1$ y $s2$ se intersecten es que exista algún x para el cuál $s1$ y $s2$ sean consecutivos en el orden $>_x$. Esto sugiere inmediatamente que la secuencia de las intersecciones del conjunto de segmentos con la línea vertical contiene toda la información relevante a la intersección de los segmentos de línea.

La estructura $\mathcal{L}$ que instrumenta la línea de barrido debe soportar las siguientes operaciones:

1. $INSERTA(s, \mathcal{L})$. Inserta el segmento s en el orden mantenido por $\mathcal{L}$.
2. $BORRA(s, \mathcal{L})$. Borra el segmento s de $\mathcal{L}$.
3. $ARRIBA(s, \mathcal{L})$. Regresa el nombre del segmento inmediatamente arriba de s en el orden $\mathcal{L}$.
4. $ABAJO(s, \mathcal{L})$. Regresa el nombre del segmento inmediatamente abajo de s en el orden $\mathcal{L}$.

De manera análoga, las operaciones de la estructura $\mathcal{E}$, (en la literatura conocida como *conjunto de eventos*) son las mismas que las de la estructura $\mathcal{L}$, por consiguiente, una estructura adecuada para almacenar tanto $\mathcal{L}$ como $\mathcal{E}$ son los árboles balanceados de búsqueda (árboles AVL [31], árboles 2-3 [31] o árboles rojo-negro [31]), ya que las operaciones que se han enunciado pueden ser efectuadas en tiempo logarítmico con respecto al número de datos que contiene.

Como el movimiento de la línea de barrido requiere tantos pasos como elementos tenga S y en cada uno de ellos se utilizan un número constante de veces las estructuras $\mathcal{L}$ y $\mathcal{E}$, concluimos que la complejidad del proceso es $O(NlogN)$ donde $n = |S|$.

El que todas las intersecciones sean localizadas se sigue de la observación de que sólo segmentos adyacentes en la línea de barrido pueden intersectarse y de que todas las adyacencias son examinadas al menos una vez.

3.2.2 Eficiencia del algoritmo

La línea 1 es ejecutada en tiempo $O(NlgN)$. El número de veces que la línea 3 es ejecutada es exactamente $2N + K$. Ya que el orden total $\mathcal{L}$ no puede contener más de N segmentos, cada operación en $\mathcal{L}$ puede ser efectuada en tiempo O(lgN). El costo de las operaciones sobre $\mathcal{E}$ es $O(lg[2N + k]) = O(lgN)$ (ya que $K \leq N^2$). Así, tenemos un costo de O(lg N) en cada una de las O(N + K) iteraciones de la línea 3, tal que el tiempo de ejecución del algoritmo es $O(NlgN + KlgN)$. Nótese que si K es muy cercana a N^2 entonces el tiempo de ejecución del algoritmo es mayor que $O(N^2)$, siendo está la complejidad del algoritmo "burdo" que checa todas las $\begin{pmatrix} N \\ 2 \end{pmatrix}$ posibles intersecciones.

3.2.3 Desventaja del algoritmo

Como mencionamos anteriormente, el algoritmo de Bentley/Ottmann presenta problemas de sensibilidad numérica ocasionados por errores de redondeo al representar valores en punto flotante en la computadora. En el siguiente ejemplo, se muestra cómo una imprecisión aritmética de este tipo hace que el algoritmo de Bentley/Ottmann no funcione adecuadamente.

Supongamos que se dispone del conjunto de segmentos que se muestra en la Figura 3.3. Aparentemente, el punto $p3$ (punto de intersección entre $s1$ y $s3$) y $p4$ (punto de intersección entre $s2$ y $s3$) tienen la misma abscisa. Sin embargo, si amplificamos adecuadamente la Figura, nos daremos cuenta que la abscisa de $p3$ es menor que la abscisa de $p4$, tal como puede apreciarse en la Figura 3.4.

Debido a que la distancia infinitesimal de la la abscisa de $p3$ con la abscisa de $p4$ es muy pequeña, puede ocurrir que una comparación aritmética en la computadora indique erróneamente que la abscisa de $p3$ es mayor que la de $p4$!. Si aplicamos el algoritmo de Bentley/Ottmann bajo esta situación veremos que no se respeta el orden de los segmentos de línea que debe mantener la línea de barrido, ocasionando que el algoritmo falle al no detectar el punto $p5$ (punto de intersección entre $s2$ y $s4$).

El proceso que sigue el método de Bentley/Ottmann en la Figura 3.4 es el siguiente:

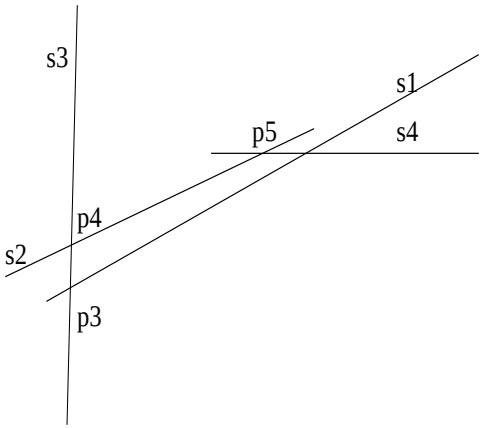


Figura 3.3: Figura para ejemplificar un fallo por redondeo.

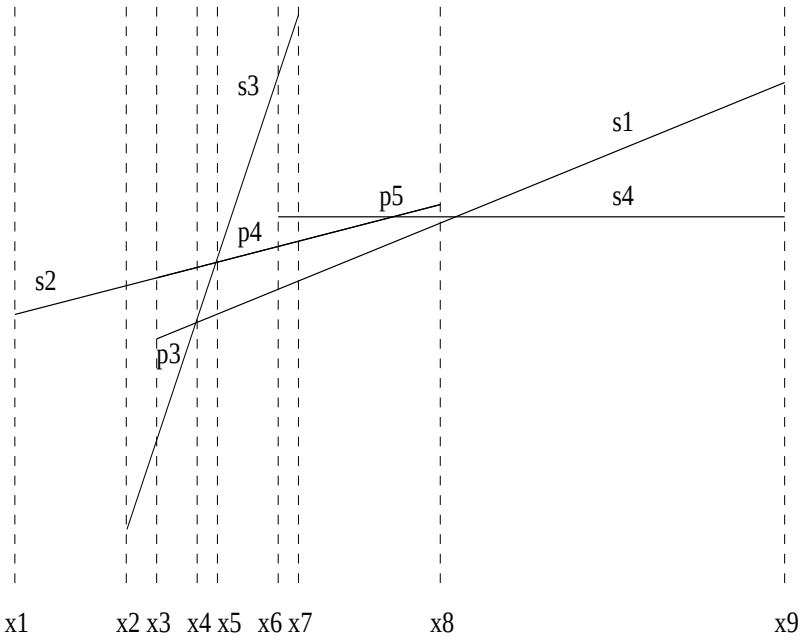


Figura 3.4: Si erróneamente se detecta primero la abscisa de $p4$ y luego la de $p3$, entonces el método de Bentley/Ottmann fallará en la detección del punto $p5$. Las líneas punteadas representan las posiciones que toma la línea de barrido al moverse de izquierda a derecha.

1. Inicialmente, la línea de barrido se encuentra en la abscisa $x1$, por lo que el segmento $s2$ es insertado en el orden mantenido por $\mathcal{L}$. Como ninguna intersección entre segmentos es detectada en este momento, no hay abscisa por agregar a $\mathcal{E}$ y sólo se elimina el elemento procesado ($x1$), por lo que $\mathcal{E}$ queda como sigue:

$$\mathcal{E} \leftarrow \{x2, x3, x6, x7, x8, x9\}$$

2. A continuación, la línea de barrido se encuentra en la abscisa $x2$, por lo que el segmento $s3$ es insertado en $\mathcal{L}$ obteniéndose la siguiente relación:

$$s2 >_{x2} s3$$

   Además, en este momento es detectado el punto de intersección $p4$, por lo que su abscisa, $x5$, es agregada a $\mathcal{E}$ y $x2$ es eliminada de $\mathcal{E}$ quedando como sigue:

$$\mathcal{E} \leftarrow \{x3, x5, x6, x7, x8, x9\}$$

3. Posteriormente, la línea de barrido se encuentra en $x3$, insertándose el segmento $s1$ en $\mathcal{L}$, por lo que la nueva relación de orden es:

$$s2 >_{x3} s3, s1 >_{x3} s3, s2 >_{x3} s1$$

   Además, el punto de intersección, $p3$, es detectado y su abscisa, $x4$, insertada en $\mathcal{E}$. Como la distancia entre $x4$ y $x5$ es muy pequeña (dentro del rango de precisión de la aritmética de punto flotante) el programa encuentra, erróneamente, que $x4 > x5$ (cuando debería ser lo contrario), por lo que el orden de $\mathcal{E}$ queda de la siguiente forma:

$$\mathcal{E} \leftarrow \{x5, x4, x6, x7, x8, x9\}$$

4. Posteriormente, la línea de barrido se encuentra en $x5$, que corresponde a la abscisa del punto de intersección $p4$, por lo que los segmentos involucrados, $s2$ y $s3$, se borran del diccionario que mantiene la línea de barrido y se vuelvan a insertar, ordenándolos con respecto a la abscisa $x5$, por lo que $\mathcal{L}$ queda como sigue:

$$s3 >_{x5} s1, s2 >_{x5} s1, s3 >_{x5} s2$$

   Nótese que el orden mantenido por $\mathcal{L}$ con respecto a la abscisa $x5$ es correcto. Por otro lado, se borra el evento procesado, $x5$, de $\mathcal{E}$ quedando como sigue:

$$\mathcal{E} \leftarrow \{x4, x6, x7, x8, x9\}$$

5. A continuación la línea de barrido alcanza el siguiente evento de $\mathcal{E}$, $x4$, que corresponde al punto de intersección $p3$ por lo que sus segmentos involucrados, $s1$ y $s3$, se borran del diccionario mantenido por la línea de barrido y se vuelven a insertar, pero ahora ordenados con respecto a la abscisa $x4$, por lo que $\mathcal{L}$ queda como sigue:

$$s3 >_{x4} s1, s2 >_{x4} s1, s2 >_{x4} s3$$

Borrando el evento procesado, $x4$, de $\mathcal{E}$ queda como sigue:

$$\mathcal{E} \leftarrow \{x6, x7, x8, x9\}$$

6. Se procesa el siguiente evento de $\mathcal{E}$, $x6$, y se encuentra que corresponde al extremo izquierdo del segmento $s4$, por lo que es insertado en el diccionario mantenido por $\mathcal{L}$. Pero como el orden mantenido por $\mathcal{L}$ es incorrecto, no es claro donde quedará insertado el segmento $s4$ (es como si se insertará el número 5 en la secuencia de números 2, 9, 7). Puede quedar insertado antes del segmento $s1$, entre los segmentos $s1$ y $s3$, entre los segmentos $s3$ y $s2$ o después del segmento $s3$. Si, por ejemplo, es insertado después del segmento $s3$, entonces el algoritmo de Bentley/Ottmann buscará posibles intersecciones entre $s4$ y los segmentos adyacentes a él. Así, buscará intersección entre el segmento $s4$ y $s3$. Como no existe intersección, no hay ningún evento por insertar en la estructura $\mathcal{E}$, por lo que el algoritmo de Bentley/Ottmann continua procesando sus eventos restantes: $x7$, $x8$ y $x9$ sin haber detectado la intersección $p5$.

   Así, podemos ver que debido a un error de comparación en la aritmética de punto flotante, el punto de intersección $p5$ no pudo ser detectado lo que provoca que el algoritmo de Bentley/Ottmann no funcione de manera adecuada.

Sumario

El método de Bentley/Ottmann es uno de los métodos clásicos de la geometría computacional para resolver el problema de las intersecciones de segmentos de línea sobre el plano de manera óptima. Sin embargo, el método tiene la desventaja de depender de la correcta aritmética de punto flotante que pueda realizar la computadora y, si un cálculo o comparación aritmética es incorrecta, lo más seguro es que el método no produzca los resultados deseados.

Desde un punto de vista formal el método de Bentley/Ottmann es correcto, y se puede instrumentar adecuadamente si se utiliza una aritmética distinta. Una alternativa podría ser programarlo en un lenguaje como Scheme (ver [13]), ya que este lenguaje permite una representación simbólica exacta de los valores numéricos.

En el próximo capítulo se discute un método alternativo que permite eliminar los problemas de inestabilidad numérica presentes en el método de Bentley/Ottmann.

Capítulo 4

Método de Edelsbrunner

4.1 Introducción

En la sección 3.2 presentamos el método óptimo registrado en la literatura para encontrar las intersecciones presentes en un conjunto dado de segmentos de línea sobre el plano. Probamos que este método tiene fuertes problemas de sensibilidad numérica, que, bajo una aritmética de punto flotante son insalvables. Esto no significa que el método de Bentley/Ottmann sea inadecuado, simplemente quiere decir que para instrumentarlo de manera correcta es necesario utilizar aritmética racional. En Internet hay varias bibliotecas que permiten desarrollar programas bajo este modelo aritmético (ver por ejemplo [20, 15]). Nosotros hemos considerado, que dadas la naturaleza y magnitud de las coordenadas en los segmentos, la aritmética racional obliga a que la representación exacta de algunos números en el método de Bentley/Ottmann requieran una gran cantidad de memoria. Por esta razón, hemos decidido instrumentar un método diferente para encontrar las intersecciones que nos interesan, nos hemos basado en el método de Edelsbrunner, el cual originalmente fue diseñado para encontrar las intersecciones presentes en un conjunto dado de rectángulos sobre el plano.

Así, el método final para determinar intersecciones que hemos programado, es el resultante de una extensión relativamente simple al método de Edelsbrunner [28]. Si bien, este método resuelve el problema planteado sin problemas de sensibilidad numérica, también es cierto que, como probamos al final del capítulo, la complejidad de la extensión tiene un comportamiento cuadrático respecto al número de segmentos a intersectar. Sin embargo, nuestras observaciones experimentales nos han demostrado que el comportamiento promedio es tan bueno como el método de Bentley/Ottmann.

4.2 Método de Edelsbrunner

Dados N rectángulos $R_1, \ldots, R_N$ en el plano, el método de Edelsbrunner encuentra los s pares de intersecciones entre ellos en un tiempo $\theta(NlogN + s)$.

Edelsbrunner resolvió este problema empleando una nueva estructura de datos que denominó *árbol de intervalos*. Si $[b^{(i)}, e^{(i)}]$ denota el intervalo en y del rectángulo R_i, sea $(y_1, y_2, ..., y_{2N})$ la secuencia ordenada de los extremos de los N intervalos. El *árbol de in-*

tervalos contiene dos estructuras, la estructura *primaria* definida estáticamente por una secuencia dada de puntos (en nuestro caso, la secuencia $(y_1, y_2, ..., y_{2N})$), y una estructura *secundaria* que puede almacenar un conjunto arbitrario de intervalos cuyos extremos están en el conjunto $\{y_1, y_2, ...y_{2N}\}$. Así, el *árbol de intervalos* T para la secuencia $(y_1, y_2, ..., y_{2N})$ y un conjunto de intervalos $I \subseteq [b^{(i)}, e^{(i)}]|i = 1, ..., N$ se define como sigue:

1. La raíz w de T tiene un discriminante $\delta(w) = (y_N + y_{N+1})/2$ y dos apuntadores a listas secundarias denotadas por $\mathcal{L}(w)$ y $\mathcal{R}(w)$. La lista $\mathcal{L}(w)$ ($\mathcal{R}(w)$) está formada por los intervalos en I que contienen al punto $\delta(w)$ en orden ascendente (descendente) con respecto a su extremo izquierdo (derecho).

2. El subárbol izquierdo de w es el árbol de intervalos para la secuencia $(y_1, y_2, ..., y_N)$ y el subconjunto $I_L \subseteq I$ de los intervalos cuyo extremo derecho es menor que $\delta(w)$. Análogamente, se define el subárbol derecho de w.

3. La estructura primaria del árbol de segmentos es T mientras que la estructura secundaria son la listas $\mathcal{L}(v)$ y $\mathcal{R}(v)$, apuntadas por los nodos en T.

4. Las hojas de T contienen árboles de los intervalos (que degeneran en un solo punto) asociados a $y_1, y_2, \ldots, y_N$.

5. Cada nodo en T se clasifica como *activo* o *inactivo*. Una nodo es activo si sus listas secundarias no están vacías o contiene nodos activos.

Tanto la estructura primaria como cada una de las listas en la estructura secundaria son árboles binarios cuya profundidad es $O(logN)$. Para insertar el intervalo $[b, e]$ se efectúa un recorrido desde la raíz hasta el primer nodo v^* de T tal que $b \leq \delta(v^*) \leq e$ (nodo de bifurcación): en ese nodo se inserta el intervalo $[b, e]$ en $\mathcal{L}(v^*)$ ordenado ascendentemente por b y en $\mathcal{R}(v^*)$ ordenado descendentemente por e. Es un ejercicio relativamente simple ver como las inserciones y borrados en esta estructura pueden ser llevados a cabo en tiempo $O(logN)$.

La búsqueda para decidir qué intervalos intersecta un intervalo dado $[b, e]$ en el árbol de segmentos, consiste en trazar una trayectoria (posiblemente vacía) desde la raíz hasta el primer nodo v^* (nodo de bifurcación) tal que $b \leq \delta(v^*) \leq e$. Enseguida, se trazan dos trayectorias divergentes desde v^* a las hojas respectivamente, asociadas con los valores de b y e (ver Figura 4.1). Supongamos que P_{IN} representa a la secuencia de nodos desde la raíz hasta el nodo v^*, P_L representa a la secuencia de nodos desde v^* hasta la hoja b, y P_R representa la secuencia de nodos desde v^* a e. Si v representa un nodo en el árbol de segmentos que se visita durante el proceso de descenso entonces se puede presentar una de las siguientes condiciones:

1. El nodo $v \in P_{IN}$. En este caso, $[b, e]$ cae ya sea a la izquierda o derecha de $\delta(v)$. Sin perdida de generalidad, se puede asumir que $e < \delta(v)$. Antes de proceder se debe checar si $[b, e]$ intersecta algún intervalo contenido en $\mathcal{L}(v)$. Esto se hace visitando cada intervalo $[b^{(i)}, e^{(i)}]$ en $\mathcal{L}$ y detectando si $b^{(i)} \leq e$, de ser así se reporta la intersección,

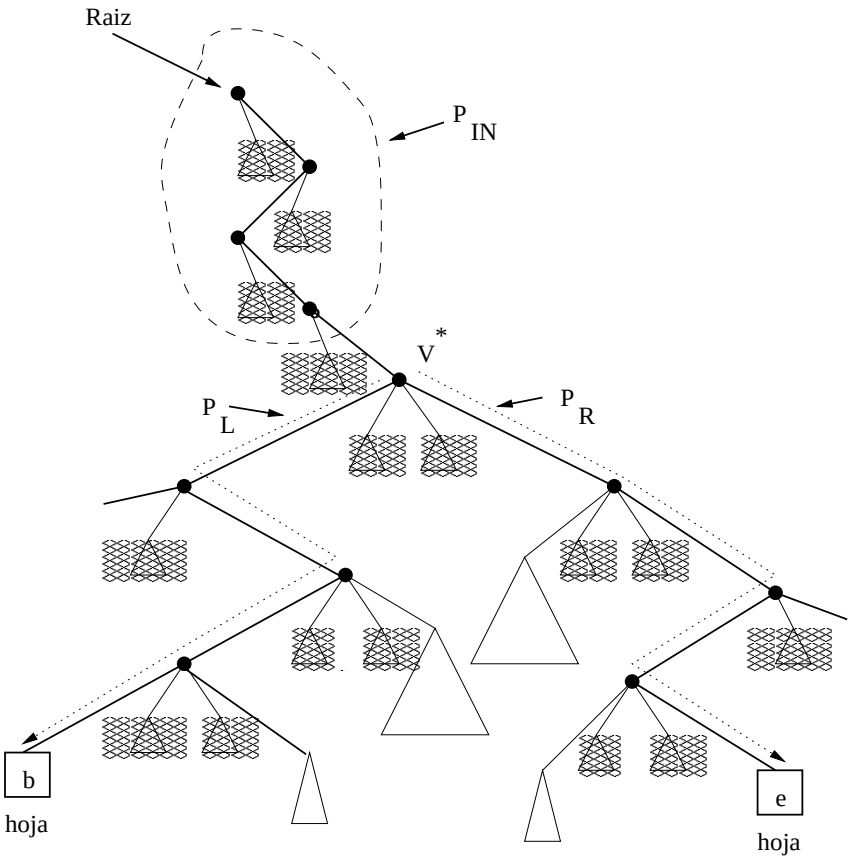


Figura 4.1: Proceso de búsqueda del intervalo $[b, e]$ en el árbol de intervalos. Los triángulos sombreados son listas secundarias y los no sombreados son sub-árboles.

el recorrido termina cuando la lista se acaba o la condición deja de cumplirse. Globalmente, el tiempo necesario para hacer estos recorridos es *proporcional* al número de intersecciones reportadas por lo que concluimos que el proceso es óptimo.

2. El nodo $v \in P_L$ (de manera análoga si $v \in P_R$). Si $\delta(v) \leq b$, entonces se recorre la lista derecha $\mathcal{R}(v)$ reportando los intervalos que contiene hasta alcanzar uno que no intersecte a $[b, e]$, el proceso es similar al explicado en el caso anterior.

   Si $b < \delta(v)$, entonces se sabe que $[b, e]$ intersecta no sólo a todos los intervalos asignados a v, sino que también a todos los intervalos asignados a los nodos en el subárbol derecho de v. El primer conjunto es obtenido reportando todos los intervalos en $\mathcal{R}(v)$. El segundo conjunto involucra reportar todos los intervalos en el subárbol derecho.

4.3 Ejemplo de operación del método de Edelsbrunner

En el ejemplo que sigue se muestra cómo crear el árbol de intervalos T a partir del conjunto de rectángulos mostrados en la Figura 4.2. Podemos ver que hay diez posibles valores para las ordenadas de las esquinas de estos rectángulos, así que el árbol de intervalos tendrá $2(10) - 1 = 19$ nodos (diez hojas y nueve nodos internos). En la Figura 4.3 puede verse el árbol de intervalos asociado al conjunto de nuestro ejemplo, cada uno de sus nodos contiene la siguiente información: el número de nodo (en la parte superior), el discriminante (en la parte inferior) y los apuntadores a las listas $\mathcal{L}$ y $\mathcal{R}$.

El método de Edelsbrunner opera de la siguiente forma sobre el conjunto de rectángulos de la Figura 4.2.

1. Para encontrar los rectángulos que se intersectan en el eje X se emplea la *técnica de barrido en el plano* que consiste en recorrer, de izquierda a derecha, los extremos de los rectángulos. Cuando el extremo izquierdo de un rectángulo es alcanzado se inserta en la línea de barrido y cuando el extremo derecho es alcanzado se borra. De esta manera, la línea de barrido sólo contiene rectángulos cuyas proyecciones sobre el eje X se intersectan.

2. Las intersecciones de un rectángulo dado con los restantes se encuentran de la siguiente manera. Si la línea de barrido alcanza el extremo izquierdo del rectángulo R_5 (mostrada con línea punteada en la Figura 4.2), entonces se inserta el intervalo $[3, 4]$ en el árbol de intervalos. Por supuesto, los intervalos en y asociados a los rectángulos R_1, R_2, R_3, R_4, R_6, R_7 y R_8 fueron insertados previamente (ver apuntadores a $\mathcal{L}$ y $\mathcal{R}$ en la Figura 4.3).

   Es fácil ver que el intervalo $[3, 4]$ tiene que ser insertado en el nodo 1 (nodo V^*), debido a que este es el primer nodo que cumple la condición $3 \leq \delta(1) = 3.5 \leq 4$. De esta inserción encontramos que $P_{IN} = \{0\}$, $P_L = \{3, 8, 18\}$ y $P_R = \{4, 9\}$.

   Para encontrar las intersecciones entre el intervalo $[3, 4]$ y los restantes se procede de la siguiente forma:

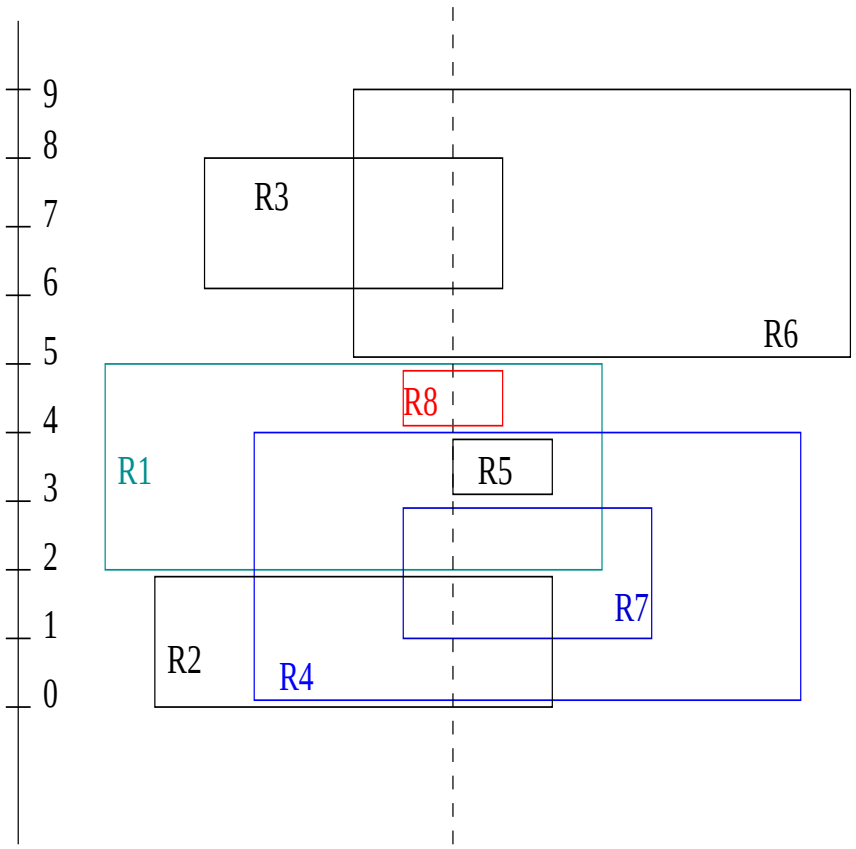


Figura 4.2: Conjunto de rectángulos. Por comodidad de visualización, algunas ordenadas se han desplazado ligeramente para evitar que se encimen los trazos.

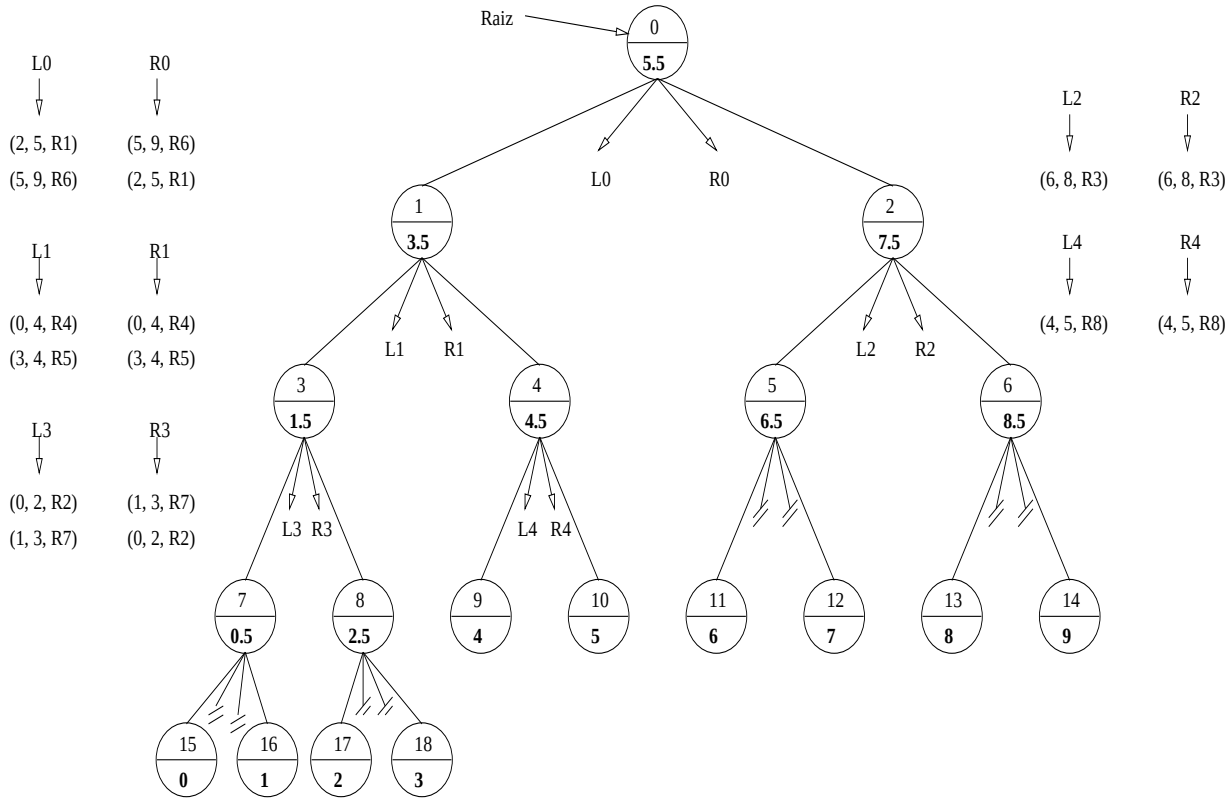


Figura 4.3: Árbol de intervalos asociado a la Figura 4.2. Para cada nodo, en la parte superior se muestra su número, en la parte inferior su discriminante y los apuntadores a $\mathcal{L}$ y $\mathcal{R}$. Los intervalos apuntados por $\mathcal{L}$ ($\mathcal{R}$) son ordenados ascendentemente (descendentemente) por su extremo izquierdo (derecho). Para cada intervalo en $\mathcal{L}$ o $\mathcal{R}$ también se guarda información de su rectángulo asociado.

(a) **Para los nodos en** P_{IN}. Sólo hay un nodo y está a la derecha del intervalo $[3, 4]$, por lo que si el extremo derecho de este intervalo es mayor o igual que el extremo izquierdo de los intervalos apuntados por $\mathcal{L}$, entonces los intersectará. Así encontramos que el intervalo $[3, 4]$ intersecta al intervalo $[2, 5]$, lo que significa que el rectángulo R_5 intersecta al rectángulo R_1.

(b) **Para los nodos en** P_L. Para el nodo 3 tenemos que sólo el intervalo $[1, 3]$ intersecta al $[3, 4]$, por lo que el rectángulo R_7 intersecta al rectángulo R_5. Como los nodos 8 y 18 no contienen intervalos en sus listas secundarias, no hay más que hacer.

(c) **Para los nodos en** P_R. Sólo el nodo 4 tiene un intervalo asociado y encontramos que intersecta al intervalos $[3, 4]$, por lo que el rectángulo R_8 intersecta al rectángulo R_5.

El intervalo $[3, 4]$ intersecta a todos los intervalos en el nodo 1, por lo que el rectángulo R_5 intersecta al rectángulo R_4.

Como podemos ver, tanto el método de Edelsbrunner como el de Bentley/Ottmann (sección 3.2) están basados en el mismo paradigma: utilizan la *técnica de la línea de barrido* y el *conjunto de eventos* para dar solución al problema de las intersecciones de segmentos de línea sobre el plano. La técnica de la línea de barrido permite guardar una relación de orden entre objetos geométricos (segmentos de línea en el método de Bentley/Ottmann y rectángulos en el método de Edelsbrunner) que dependerá de su ubicación dada por el conjunto de eventos.

4.4 Extensión al método de Edelsbrunner

Ahora presentamos cómo hemos extendido el método de Edelsbrunner para resolver el problema de las intersecciones entre segmentos de línea en el plano, evitando los problemas aritméticos del Método de Bentley/Ottmann. La clave de esta extensión es que no necesita tomar decisiones a partir de resultados de operaciones de punto flotante.

En esta extensión, primero se asocia el mínimo rectángulo que contiene a cada uno de los segmentos de línea, enseguida, se aplica el método de Edelsbrunner para encontrar todos los pares de intersecciones entre rectángulos, y finalmente para cada par de rectángulos que se intersecta se prueba si sus segmentos de línea asociados también lo hacen. De esta manera, se reportan los pares de intersecciones entre segmentos de línea en el mismo tiempo que el método de Edelsbrunner.

Los cálculos aritméticos no intervienen en la toma de decisiones pues el método de Edelsbrunner no lo necesita. Hasta el momento en que el método de Edelsbrunner detecta la intersección entre dos rectángulos se calcula la intersección de los segmentos asociados. De esta forma, resolvemos el problema de la sensibilidad numérica pero perdemos eficiencia porque muchas de las intersecciones entre rectángulos detectadas no corresponden a intersecciones entre segmentos.

Vale la pena analizar las expresiones para la complejidad temporal de ambos métodos, en el caso de Bentley/Ottmann se requiere un tiempo $O((N + K)logN)$, mientras que el de Edelsbrunner lo hace en tiempo $O(NlogN+s)$. En el primer caso, K representa el número de

intersecciones entre segmentos de línea, mientras que en el segundo, s representa el número de intersecciones entre rectángulos. En general $s \geq K$.

Aparentemente, estos comportamientos temporales son muy semejantes. Sin embargo, podemos encontrar "configuraciones" de segmentos de línea en las que el método de Bentley/Ottmann aventaja al método de Edelsbrunner. Por ejemplo, en el inciso a) de la Figura 4.4 el factor s es $O(N^2)$ para Edelsbrunner (ya que todos los rectángulos se intersectan entre sí) y $K = 0$ para el método de Bentley/Ottmann. Esto equivale a que el método de Edelsbrunner tenga un comportamiento $O(N^2)$ y Bentley/Ottmann $O(NlogN)$. Es fácil ver que en el inciso b) de la misma figura ocurre lo mismo. Así, el peor caso de la extensión al método de Edelsbrunner puede tener un comportamiento $O(N^2)$.

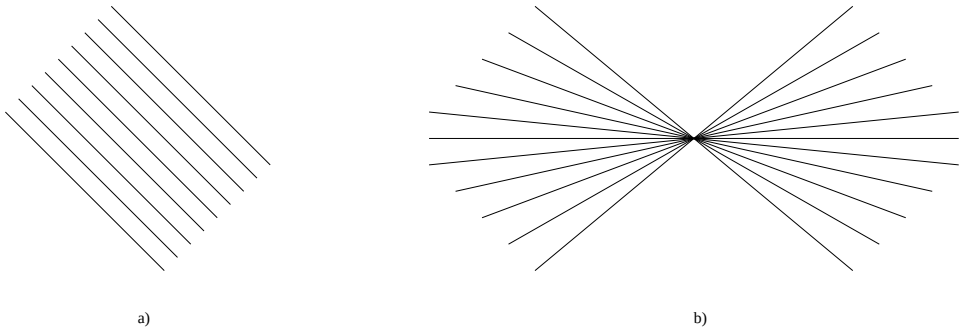


Figura 4.4: En ambas figuras, el comportamiento temporal de Bentley/Ottmann es $O(NlogN)$ y de Edelsbrunner es $O(N^2)$.

Sin embargo, también existen configuraciones en las que el método de Edelsbrunner supera al método de Bentley/Ottmann. Por ejemplo, cuando $K = O(N^2)$, s también tiene un comportamiento cuadrático (no puede ser peor), tenemos que la complejidad temporal para el método de Edelsbrunner es $O(N^2)$ mientras que para el de Bentley/Ottmann es $O(N^2logN)$ (ver Figura 4.5).

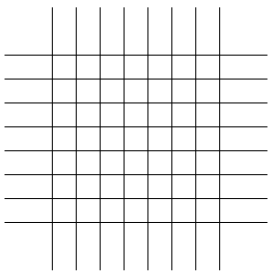


Figura 4.5: El comportamiento temporal de Bentley/Ottmann es $O(N^2logN)$ y de Edelsbrunner es $O(N^2)$.

Desde el punto de vista del comportamiento promedio no hicimos un análisis preciso de la complejidad, porque ésta depende de la estructura de los conjuntos de segmentos rectilíneos de la entrada, pero sí hicimos una comparación experimental aplicando ambos algoritmos a

grades mapas (mayores a 10,000 segmentos de línea). Encontramos, en todos los casos, un comportamiento temporal muy cercano en ambos métodos, lo cual es simple de justificar: Los dos métodos se basan en el mismo paradigma (línea de barrido, conjunto de eventos, etc.) y los rectángulos que contienen a los segmentos son demasiado pequeños, de hecho el número de rectángulos que intersecta un rectángulo dado es, en promedio, un número pequeño. Así, de estos dos hechos es inmediato que ambos métodos presentan una complejidad $O(NlogN)$ en promedio.

Los fuertes problemas de sensibilidad numérica del método de Bentley/Ottmann no significan que tenga que ser desechado como opción para resolver el problema de las intersecciones de segmentos de línea. En realidad, el método puede ser rescatado si se lleva a cabo la implementación de una aritmética robusta para garantizar la exactitud en los cálculos y comparaciones aritméticas. Hay varias alternativas: aritmética racional ([20]), aritmética con precisión infinita (como lo hace matemática, [15]), etc.

Sumario

La extensión al método de Edelsbrunner encuentra las intersecciones de un conjunto dado de segmentos rectilíneos, pero sin los problemas de sensibilidad numérica que presentó el método de Bentley/Ottmann. Aunque en principio, es un método de complejidad temporal cuadrática (en el peor caso), observamos experimentalmente que en promedio su complejidad temporal es equiparable a la del método de Bentley/Ottmann. En el disco compacto anexo al documento de tesis se muestra el código en lenguaje C del método de Edelsbrunner.

Capítulo 5

Representación formal de mapas en el plano y su construcción a partir del conjunto de segmentos que lo define

En este Capítulo se define de manera rigurosa el concepto de "mapa" a partir de la idea de encaje de una gráfica en el plano. Después se introduce la "Lista de Aristas Doblemente Conectadas" (LADC), que sirve para representar encajes en la memoria de una computadora. Finalmente, se muestra un algoritmo para construir la LADC de un conjunto de segmentos en el plano. La LADC es el medio básico para representar mapas en nuestro núcleo.

5.1 Encajes de gráficas en el plano

Una *gráfica* $G = (V, E)$ (ver [30], [6], [7] y [32]) es un par ordenado de conjuntos disjuntos V, E tal que $E \subseteq V \times V$ y $V \neq \emptyset$. El conjunto V es el conjunto de *vértices* de G y E es el conjunto de *aristas*. Una arista une a los vértices μ y ν y se denota por (μ, ν).

Un *encaje planar* (ver [35] y [32]) Ψ se define como una triada (Σ, U, V) donde Σ es una superficie, U es un subconjunto cerrado (en el sentido topológico) de Σ y V es un subconjunto finito de U tal que $C(U-V)$ es un conjunto finito de copias homomórficas del intervalo $(0, 1)$. Denotamos por $\Sigma(\Psi)$ la superficie en la cual se encaja a Ψ, por $V(\Psi)$ el conjunto de vértices V de Ψ, por $E(\Psi)$ el conjunto de aristas $C(U - V)$ de Ψ y por $F(\Psi)$ el conjunto de caras $C(U - V)$ de Ψ. Así que la superficie $\Sigma(\Psi) = \Sigma$ de Ψ es particionada en vértices, aristas y caras, respectivamente. La gráfica $G(\Psi)$ de Ψ es la gráfica con el mismo conjunto de vértices y aristas que Ψ, en la cual cada arista es incidente con uno o dos vértices de su cerradura.

Una gráfica es planar si admite un encaje planar y una LADC (Lista de Aristas Doblemente Conectadas) es una estructura de datos para representar una gráfica planar.

Desde el punto de vista formal un *mapa* es un encaje planar. La gráfica de este encaje contiene al conjunto de puntos en el mapa donde se intersectan dos o más curvas y dos de estos vértices se unen por una arista siempre que en el mapa exista un trazo continuo que los une.

La naturaleza de la definición que se ha dado permite definir mapas no sólo en el plano (o la esfera que salvo por el punto en el infinito es equivalente al plano), sino también

en cualquier otra superficie orientable [17], como el toro, doble toro, etc. De hecho, la formulación combinatoria que se hace a continuación es un caso particular de una estrategia de representación de mapas conocida como *mapas combinatorios* debida a W. Tutte [34].

5.2 Lista de aristas doblemente conectadas (LADC)

La LADC de un encaje planar $\psi = (\Sigma, U, V)$, denotada por $LADC(\psi)$ es una triada (nl, v, c) donde nl, v y c son arreglos que sirven para representar, respectivamente, las aristas, los vértices y las caras de ψ. Con esta estructura es posible plantear, de manera eficiente, las operaciones típicas sobre un mapa como recorrer la frontera de una región, recorrer las aristas incidentes en un vértice, determinar cuáles son las caras o los vértices incidentes a una arista, buscar la región que contiene un punto a partir de sus coordenadas, etc. La LADC se define formalmente como sigue:

Suponemos que $V = \{v_1, ..., v_N\}$ y $E = \{e_1, ..., e_M\}$ son, respectivamente, los conjuntos de vértices y aristas de $G(\Psi)$, asimismo suponemos que las aristas en $G(\Psi)$ han sido orientadas de manera arbitraria. La componente principal en la LADC de Ψ es el *arreglo de nodos lado* (nl) que contiene $\mid E \mid$ entradas. Hay una correspondencia uno a uno entre las aristas y las entradas de este arreglo. La entrada j-ésima de nl se denota por $nl[j]$ y consiste de cuatro campos de información V_1, V_2, C_1 y C_2, y dos campos apuntador P_1 y P_2. Cuando sea necesario usaremos la notación $nl[j].C_1$, $nl[j].C_2$, etc. El significado de los campos es el siguiente. El campo V_1 contiene el origen de la arista y V_2 el fin, de acuerdo con la orientación que tenga la arista. Los campos C_1 y C_2 contienen los nombres de las caras ubicadas, respectivamente, en el lado izquierdo y derecho de la arista orientada de V_1 a V_2. El apuntador P_1 (respectivamente P_2) apunta al nodo lado de la primer arista encontrada después de que la arista (V_1, V_2) es girada en sentido contrario al de las manecillas del reloj alrededor de V_1 (respectivamente V_2). Los nombre de las caras, vértices y nodos lado son números enteros no negativos seleccionados de manera arbitraria e identifican de manera única a cada uno de estos elementos. En la Figura 5.1 se muestra el fragmento de una gráfica y en la Tabla 5.1 su LADC correspondiente.

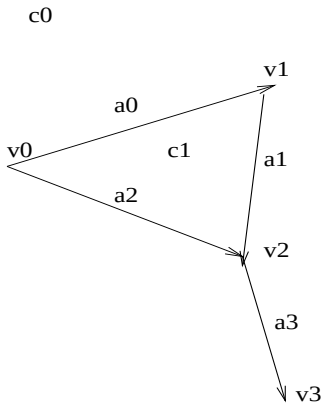


Figura 5.1: Gráfica usada para construir su LADC.

arista	V_1	V_2	C_1	C_2	P_1	P_2
a0	0	1	0	1	2	1
a1	1	2	0	1	0	2
a2	0	2	1	0	0	3
a3	2	3	0	0	1	3

Tabla 5.1: LADC correspondiente a la Figura 5.1.

5.3 Algoritmo para construir la LADC de un encaje planar

Nuestro algoritmo para construir la LADC toma como entrada los segmentos de línea que definen las aristas en el encaje planar. Suponemos que dos de estos segmentos elegidos arbitrariamente se intersectan a lo más en sus extremos. En el Capítulo 3 se ha explicado cómo transformar un conjunto de segmentos que no satisfagan esta condición.

Supongamos que Σ representa al plano y que $s_1, s_2 \ldots s_n$ son segmentos de línea en Σ. El algoritmo procede iterativamente de tal manera que en el j-ésimo paso se construye la LADC correspondiente al encaje $(\Sigma, \{s_1, \ldots, s_j\}, \bigcup_{k=1,\ldots,j}\{i(s_k), f(s_k)\})$ donde $i(s_k)$ y $f(s_k)$ representan, respectivamente, los extremos inicial y final de s_k.

De esta manera en el paso j-ésimo, el j-ésimo nodo lado queda asociado con el segmento s_j y se asignan valores a sus campos de la siguiente manera.

1. Al campo V_1 se le asigna el nombre de $i(s_j)$, este nombre se busca en un diccionario que contiene los nombres de todos los vértices que se han incorporado a la LADC hasta el paso $j-1$ y si no aparece en el diccionario, entonces se le asigna un nombre no utilizado y se guarda tal asignación en el diccionario. El campo V_2 se asigna de manera semejante.

2. Se localizan los conjuntos $\mathcal{I}_j$ y $\mathcal{F}_j$ conteniendo, respectivamente, los segmentos del encaje incidentes con $i(s_j)$ y $f(s_j)$ que ya se han incorporado a la LADC. Si $\mathcal{I}_j \neq \emptyset$ ($\mathcal{F}_j \neq \emptyset$) entonces denotamos por s_{j,i_1} y s_{j,i_2} (s_{j,f_1} y s_{j,f_2}), respectivamente, los segmentos ubicados antes y después de s_j en $\mathcal{I}_j$ ($\mathcal{F}_j$) en el orden contrario a las manecilla del reloj al girar sobre $i(s_j)$ ($f(s_j)$). Nótese que si $|\mathcal{I}_j| = 1$ entonces $s_{j,i_1} = s_{j,i_2}$ (ver Figura 5.2).

   Tenemos los siguientes casos:

   (a) Si $\mathcal{I}_j \bigcup \mathcal{F}_j = \emptyset$ entonces a los campos C_1 y C_2 se les asigna el mismo nombre de manera arbitraria (preferentemente el menor número no utilizado como nombre de cara).

   (b) Si $\mathcal{I}_j \neq \emptyset$ y $\mathcal{F}_j = \emptyset$, entonces C_1 y C_2 se asignan como el nombre de la cara común incidente a los segmentos s_{j,i_1} y s_{j,i_2}.

   (c) El caso $\mathcal{I}_j = \emptyset$ y $\mathcal{F}_j \neq \emptyset$ se trata de manera análoga.

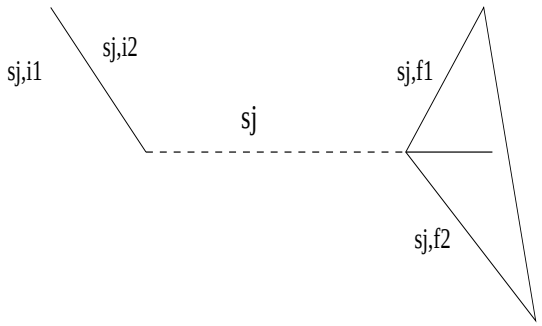
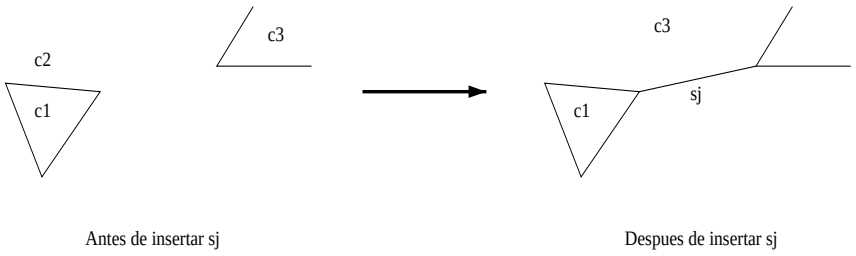


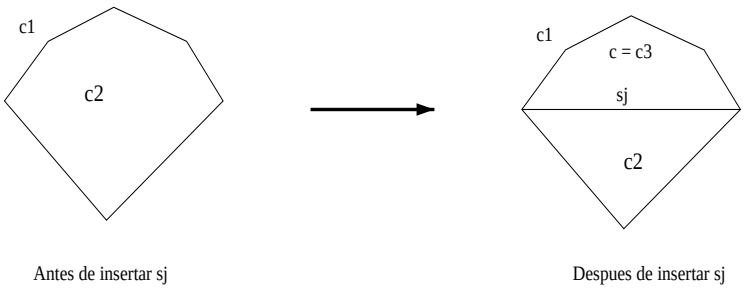
Figura 5.2: Segmentos antes y después de s_j.

(d) Si $\mathcal{I}_j \neq \emptyset$ y $\mathcal{F}_j \neq \emptyset$. Sea c_i el nombre de la cara común entre los segmentos s_{j,i_1} y s_{j,i_2} y c_f el nombre de la cara común entre los segmentos s_{j,f_1} y s_{j,f_2}. Existen dos sub-casos:

i. Si $c_i \neq c_f$ entonces se hacen los asignamientos $C_1 \leftarrow C_2 \leftarrow c_f$ y en todos los nodos lado que en su campo C_1 o C_2 tengan el valor c_i se cambia por c_f. La idea es que la cara c_f absorbe a la cara c_i (Ver inciso (a) de la Figura 5.3)



(a) ci != cf



(b) ci = cf

Figura 5.3: Reetiquetado de caras. En (a) cuando $c_i \neq c_f$ y en (b) cuando $c_i = c_f$.

ii. Si $c_i = c_f$ entonces se recorren los segmentos incidentes con la cara c_i en el sentido de las manecillas del reloj partiendo de s_{j,i_2} hasta s_{j,f_1}, en cada uno de

los nodos lado de los segmentos visitados se cambian los campos C_1 o C_2 que tengan el valor c_i por el nombre c de una cara nueva disponible. Finalmente, en el j-ésimo nodo lado se hacen los asignamientos $C_1 \leftarrow c$ y $C_2 \leftarrow c_i$. (Ver inciso (b) de la Figura 5.3)

(e) Finalmente, se hacen los asignamientos $P_1 \leftarrow$ nodo lado asociado a s_{j,i_2} y $P_2 \leftarrow$ nodo lado asociado a s_{j,f_2}.

5.4 Operaciones de navegación sobre la LADC

En general consultar un mapa involucra dos tipos de operaciones: el arribo y la navegación secuencial. Pongamos un ejemplo, supongamos que se tiene una LADC que representa al mapa de la República Mexicana y que deseamos mostrar el contorno del estado de Sonora. Lo primero que debemos hacer es arribar, de alguna manera, a una arista o un vértice incidente a la cara que representa al estado de Sonora o a la cara misma, esta es la operación de arribo. Una vez que estemos ahí debemos seguir la frontera del estado de Sonora reportando la secuencia de aristas visitadas, esta es la operación de navegación.

Las operaciones de arribo se pueden dar de varias formas, pero la más común es mediante la localización de un punto dentro de una región. En nuestro ejemplo, se podrían especificar las coordenadas geográficas de un punto dentro del estado (Por ejemplo del zócalo de Sonora) y el arribo se hace mediante un algoritmo que determine cuál es la región que contiene tal punto. Esta operación fundamental es discutida en el Capítulo 6. Por ahora nos concentraremos en las operaciones de navegación.

Las operaciones de navegación deben permitir recorrer elementos en la LADC de manera secuencial, es decir de un elemento a otro que incide en él. En nuestro ejemplo, tomábamos una arista sobre el estado de Sonora y a partir de ella recorríamos secuencialmente la frontera del estado, digamos que en el sentido en que giran las manecillas del reloj. Observemos que de la misma manera, una operación de navegación de este tipo podría involucrar fijar un vértice y encontrar todas las caras o aristas que inciden en el vértice en sentido contrario a las manecillas del reloj, esta operación se utilizó en la sección anterior para definir las aristas s_{j,i_1} y s_{j,i_2} en el algoritmo para generar la LADC. También podríamos fijar una arista y recorrer sus extremos o sus caras incidentes. Estos tipos de recorridos pueden ser registrados mediante el uso de un "cursor".

Para explicar la idea de "cursor" en mapas de una manera sencilla, podemos recurrir a la analogía que se tiene en el manejo de un editor de texto. Para movernos de una posición específica en el editor a otra lo hacemos moviendo un cursor, que se desplaza a la izquierda, derecha, arriba o abajo, siempre hacia posiciones vecinas del caracter donde estamos ubicados. Este mismo concepto, de una manera natural, puede ser extendido para hacer navegaciones sobre un mapa, y al igual que en el caso de los editores de texto debemos empezar por definir el concepto de "cursor".

En el editor de texto, la estructura del espacio de edición se reduce a las posiciones de la pantalla que bien se pueden mirar como celdas en una matriz rectangular, así pues la posición del cursor en un momento específico queda dada por el renglón y columna donde está ubicado. A diferencia, en un encaje planar intervienen tres elementos estructurales: las

caras, las aristas y los vértices. En la LADC un cursor se define por una *cara actual*, un *vértice actual* y una *arista actual*, que son elementos del encaje y satisfacen lo siguiente:

C1 La cara actual, el vértice actual y la arista actual son mutuamente incidentes.

C2 Si recorremos la arista actual del vértice actual al otro vértice (de la arista actual), encontraremos que la cara actual se encuentra a mano derecha.

En términos del cursor así definido, la navegación en la LADC se especifica a partir del cursor fijando una de las partes del cursor y moviendo las otras dos partes ya sea en el sentido en el que giran las manecillas del reloj o el sentido inverso.

En lo que resta de esta sección supondremos que (nl, v, c) es una LADC, y que *a_actual*, *v_actual* y *c_actual* son los índices de la arista, el vértice y la cara actual del cursor de esta LADC. Las operaciones de navegación sobre la LADC son las siguientes:

1. **Rotación del cursor respecto a una cara:**. La rotación puede darse en el sentido de las manecillas del reloj o en el sentido contrario, así que tenemos dos operaciones:

   (a) **anterior_av_LADC**. Fija la posición de la cara *c_actual* y modifica los valores de *a_actual* y *v_actual* moviéndolos en el sentido de las manecillas del reloj con respecto a la cara *c_actual*. Aplicando esta operación al inciso (I) de la Figura 5.4 se obtiene el inciso (II) como resultado. El algoritmo de esta operación es:

   Algorithm 5.4.1 *Operación de navegación anterior_av_LADC.*

   Entrada: *Valores iniciales de a_actual, c_actual y v_actual.*

   Salida: *Valores finales de a_actual, c_actual y v_actual.*

   Método:

   ```
   1.  if(v_actual == nl[a_actual].v1
   2.      lado = nl[a_actual].p2;
   3.  else
   4.      lado = nl[a_actual].p1;
   5.  if(v_actual == nl[a_actual].v1
   6.      v_actual = nl[a_actual].v2;
   7.  else
   8.      v_actual = nl[a_actual].v1;
   9.  a_actual = lado;
   ```

   (b) **siguiente_av_LADC**. Fija la posición de la cara *c_actual* y modifica los valores de *a_actual* y *v_actual* moviéndolos en sentido contrario de las manecillas del reloj con respecto a la cara *c_actual*. Aplicando esta operación al inciso (I) de la Figura 5.4 se obtiene el inciso (III) como resultado. El algoritmo de esta operación es:

Algorithm 5.4.2 *Operación de navegación siguiente_av_LADC.*

Entrada: *Valores iniciales de a_actual, c_actual y v_actual.*

Salida: *Valores finales de a_actual, c_actual y v_actual.*

Método:

```
1.  lado = a_actual;
2.  do
3.      if(v_actual == nl[lado].v1)
4.          lado = nl[lado].p1;
5.      else
6.          lado = nl[lado].p2;
7.      if(v_actual == nl[lado].v1)
8.          if(a_actual == nl[lado].p1) fin = 1;
9.          else fin = 0;
10.     else
11.         if(a_actual == nl[lado].p2) fin = 1;
12.         else fin = 0;
13. while(!fin);
14. a_actual = lado;
15. if(v_actual == nl[lado].v1)
16.     v_actual = nl[lado].v2;
17. else
18.     v_actual = nl[lado].v1;
```

2. **Rotación del cursor respecto a un vértice**. La rotación puede darse en el sentido de las manecillas del reloj o en el sentido contrario, así que tenemos dos operaciones:

   (a) **anterior_ac_LADC**. Fija la posición del vértice *v_actual* y modifica los valores de *a_actual* y *c_actual* moviéndolos en el sentido de las manecillas del reloj con respecto al vértice *v_actual*. Aplicando esta operación al inciso (IV) de la Figura 5.4 se obtiene el inciso (I) como resultado. El algoritmo de esta operación es:

   Algorithm 5.4.3 *Operación de navegación anterior_ac_LADC.*

   Entrada: *Valores iniciales de a_actual, c_actual y v_actual.*

   Salida: *Valores finales de a_actual, c_actual y v_actual.*

   Método:

```
1.  lado = a_actual;
2.  do
3.      if(v_actual == nl[lado].v1)
4.          lado = nl[lado].p1;
5.      else
6.          lado = nl[lado].p2;
7.      if(v_actual == nl[lado].v1)
8.          if(a_actual == nl[lado].p1) fin = 1;
9.          else fin = 0;
10.     else
11.         if(a_actual == nl[lado].p2) fin = 1;
12.         else fin = 0;
```

13. **while***(!fin);*
14. *a_actual = lado;*
15. **if***(v_actual == nl[lado].v1)*
16. *c_actual = nl[lado].c2;*
17. **else**
18. *c_actual = nl[lado].c1;*

(b) **siguiente_ac_LADC**. Fija la posición del vértice *v_actual* y modifica los valores de *a_actual* y *c_actual* moviéndolos en sentido contrario de las manecillas del reloj con respecto al vértice *v_actual*. Aplicando esta operación al inciso (I) de la Figura 5.4 se obtiene el inciso (IV) como resultado. El algoritmo de esta operación es:

Algorithm 5.4.4 *Operación de navegación siguiente_ac_LADC.*

Entrada: *Valores iniciales de a_actual, c_actual y v_actual.*

Salida: *Valores finales de a_actual, c_actual y v_actual.*

Método:
1. *lado = a_actual;*
2. **if***(v_actual == nl[lado].v1)*
3. *a_actual = nl[lado].p1;*
4. **else**
5. *a_actual = nl[lado].p2;*
6. *lado = a_actual;*
7. **if***(c_actual == nl[lado].c1)*
8. *c_actual = nl[lado].c2;*
9. **else**
10. *c_actual = nl[lado].c1;*

3. **Rotación del cursor respecto a una arista**. En esta caso, la rotación sólo se da en el sentido contrario de las manecillas del reloj, así que tenemos la siguiente operación:

(a) **siguiente_cv_LADC**. Fija la posición de la arista *a_actual* y modifica los valores de *c_actual* y *v_actual* moviéndolos en sentido contrario de las manecillas del reloj con respecto a la arista *a_actual*. Aplicando esta operación al inciso (I) de la Figura 5.4 se obtiene el inciso (V) como resultado. El algoritmo de esta operación es:

Algorithm 5.4.5 *Operación de navegación siguiente_cv_LADC.*

Entrada: *Valores iniciales de a_actual, c_actual y v_actual.*

Salida: *Valores finales de a_actual, c_actual y v_actual.*

Método:
1. *lado = a_actual;*
2. **if***(c_actual == nl[lado].c1)*
3. *c_actual = nl[lado].c2;*
4. **else**
5. *c_actual = nl[lado].c1;*

```
6.   if(v_actual == nl[lado].v1)
7.       v_actual = nl[lado].v2;
8.   else
9.       v_actual = nl[lado].v1;
```

Siempre que se termina de ejecutar alguna de estas operaciones siguen siendo válidas las condiciones C1 y C2.

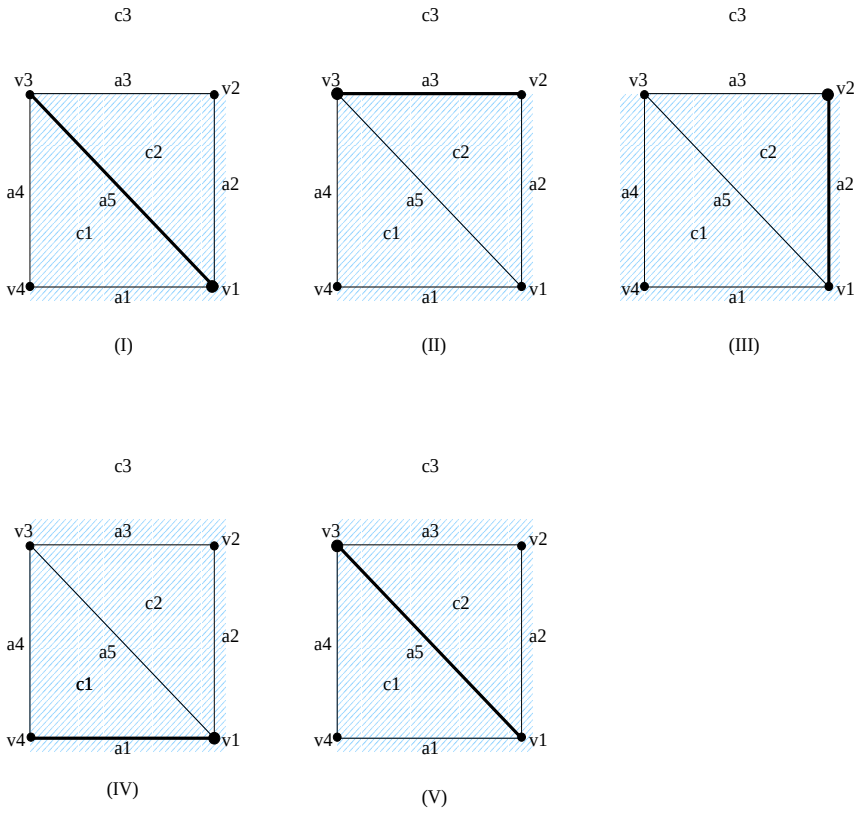


Figura 5.4: LADC empleada para mostrar las operaciones de navegación.

Este principio de movimiento de alguna manera fue introducido por Tutte (ver [34]), aunque en un contexto muy diferente al de navegación sobre el encaje. Tutte trabajó extensivamente en la formalización de encajes sobre superficies y definió el concepto de *bandera*, que es isomorfo a nuestra definición de cursor. Las banderas sirven esencialmente para determinar la orientabilidad o no orientabilidad de una superficie siguiendo trayectorias de operaciones parecidas a las que aquí hemos explicado. Nuestra contribución, esencialmente, consistió en entender el potencial de las banderas como mecanismos de navegación y su aplicación dentro de nuestro núcleo.

5.5 Superposición de las LADC que intervienen en una aplicación SIG

Al tratar de responder una consulta concreta sobre un SIG en general no se trabaja con un solo mapa, sino con varios mapas que contienen diferentes aspectos temáticos de la zona en estudio. El objetivo, normalmente, es mirar el conjunto como un todo y plantear consultas directas a ese todo en lugar de hacerlo mapa por mapa. Por ejemplo, si queremos encontrar una zona propicia para sembrar tomates y sabemos que los tomates se desarrollan mejor bajo ciertas condiciones de temperatura, pendiente y humedad del suelo, entonces es deseable plantear de una sola vez la consulta en lugar de plantearla para el mapa de temperatura, de pendiente y de humedad.

En nuestro ejemplo, si planteamos la consulta sobre el mapa de temperaturas, seremos capaces de encontrar regiones con la temperatura adecuada para la siembra del tomates, pero algunas de estas regiones contendrán sub-regiones que no satisfagan los criterios de humedad y pendiente del suelo. El mismo problema se tendría si empezáramos a construir nuestra respuesta usando los otros dos mapas. El punto crucial es que las regiones en cualquiera de los mapas originales, no satisfacen la primera forma normal de las bases de datos porque no son indivisibles, en el mapa de temperaturas, por ejemplo, cada región correspondiente a una isoterma que contiene varias pendientes con respecto al mapa topográfico. Sin embargo el proceso de normalización es evidente, bastará subdividir, hasta el nivel que sea necesario las regiones de un mapa para lograr que sus atributos (con respecto a los demás mapas) sean uniformes en cada región. El efecto final se puede describir de una manera muy sencilla, simplemente hay que "superponer" todos los mapas y generar el mapa correspondiente a la superposición.

Las caras del mapa superposición de todos los mapas que intervienen en una aplicación pertenecen a una y sólo una región de los demás mapas. Es conveniente representar esta relación de manera que no sea necesario hacer navegaciones adicionales en los mapas originales.

En la Figura 5.5 se muestran ejemplos tentativos de mapas de temperatura, pendientes y humedades del suelo. En el inciso d) de la misma figura aparece la superposición de estos mapas. Observamos que la cara C_{d1} del mapa superposición está contenida, respectivamente, en las caras C_{a1}, C_{b1} y C_{c1} de los mapas originales. En la Tabla 5.2 se muestra la relación de contención completa entre las caras del mapa superposición y los mapas originales, por comodidad solo se muestra la manera de obtener cuatro de las caras del mapa superposición. Así, la consulta "despliega todas las regiones propias para el cultivo del tomate" se reduce a encontrar las caras del mapa superposición que satisfacen las restriciones para el buen crecimiento de este producto.

Para construir la LADC de la superposición de todos los mapas se emplea el algoritmo descrito en la Sección 5.3 utilizando los segmentos de todos los mapas originales. La construcción de la relación que representa la superposición de todos los mapas se hace simplemente determinando las regiones de cada mapa que contienen a una región específica. Esto último se puede hacer de manera eficiente con el algoritmo de localización de la región que contiene a un punto dado que se describe en la Sección 6.1.

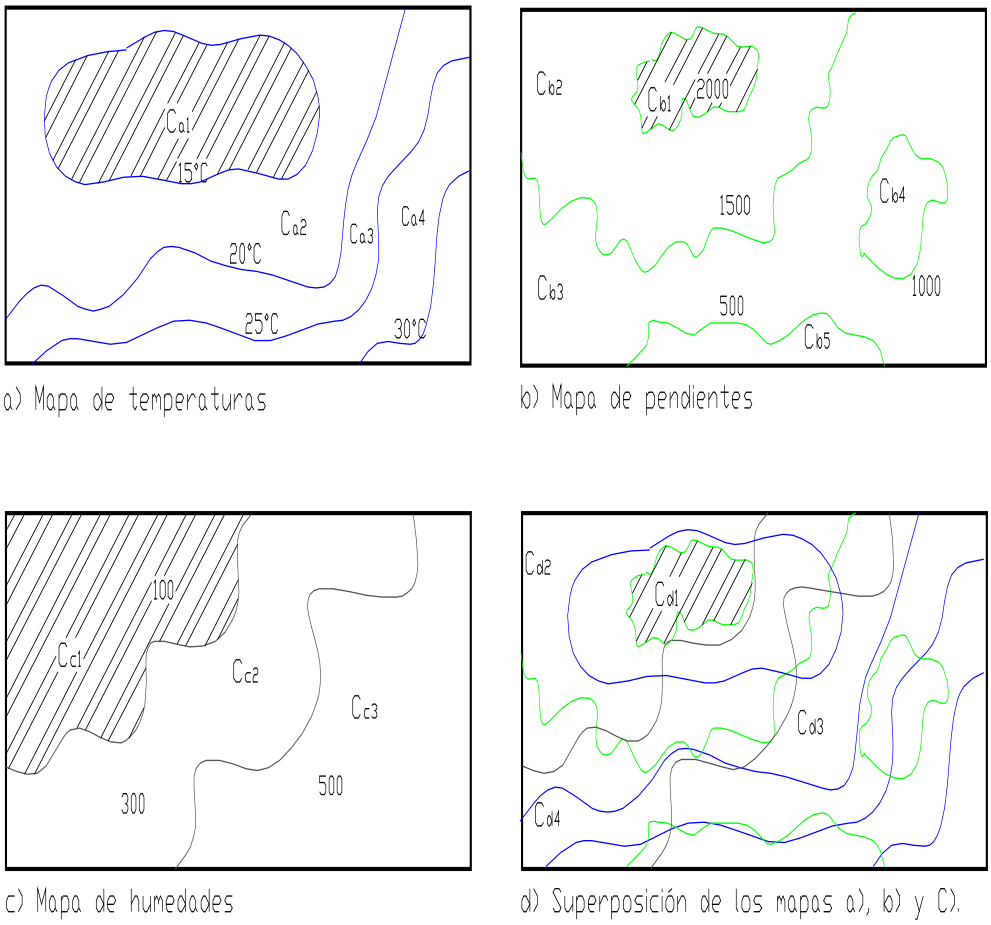


Figura 5.5: El mapa d) es la superposición de los mapas de temperaturas, pendientes y humedades.

Cara mapa superposición	Cara mapa temperaturas	Cara mapa pendientes	Cara mapa humedades
C_{d1}	C_{a1}	C_{b1}	C_{c1}
C_{d2}	C_{a2}	C_{b2}	C_{c1}
C_{d3}	C_{a2}	C_{b3}	C_{c3}
C_{d4}	C_{a3}	C_{b3}	C_{c2}

Tabla 5.2: Relación de contención entre el mapa superposición y los mapas originales.

Sumario

En este capítulo se mostró el algoritmo que permite representar mapas en la memoria de la computadora haciendo uso de la estructura de datos LADC (Lista de Aristas Doblemente Conectadas), cuya utilidad principal es la de poder plantear estrategias eficientes de búsquedas o consultas en un mapa dado.

Por otro lado, también se muestra cómo crear la LADC de la superposición de dos o mas mapas cuyo objetivo principal es poder realizar consultas más complejas en un SIG. En el disco compacto anexo al documento de tesis se muestra el código en lenguaje C del algoritmo que construye la LADC.

Capítulo 6

Algoritmo de localización de puntos en un encaje planar

En el Capítulo 5 vimos que una gráfica planar admite un encaje planar. Sin pérdida de generalidad, se asume que la gráfica planar es siempre conexa.

El problema de localizar puntos en un encaje planar consiste en determinar la región de un mapa que contiene a un punto dado. La mayoria de los algoritmos para resolver este problema subdividen las regiones de la gráfica en polígonos con el objeto de simplificar las operaciones de búsqueda. Los métodos más eficientes están basados en bisección o búsqueda binaria. La idea fundamental es crear "nuevos objetos geométricos" que permitan efectuar una búsqueda en tiempo logarítmico. El *método slab*, es una forma común de resolver este problema. El método aparece reportado en [29] pero no se indica quien es su autor. Aquí mostraremos las ideas básicas del método y algunos detalles sobre su programación.

6.1 Método slab

Sea $\psi = (\Sigma, U, V)$ un encaje planar y (x, y) un punto en el plano cuya ordenada es mayor que la menor ordenada de un vértice en V y menor que la mayor ordenada de un vértice en el mismo conjunto. La idea es encontrar cuál es la cara de ψ que contiene a (x, y), para ello suponemos que se ha construido la LADC $LADC(\psi)$ de modo que el resultado de nuestro método será el nombre de la cara que contiene a (x, y).

Supondremos, por simplicidad, que no hay dos vértices en V con la misma ordenada y que $V = \{v_1, \ldots, v_N\}$ donde los índices se han asignado de manera que la ordenada de v_i es menor que la de v_{i+1}. Para cada $i \in \{1, \ldots, N-1\}$ definimos el i-ésimo *slab* S_i de ψ como el conjunto $\{(x, y) \in \Re^2 | v_i = (x_i, y_i), v_{i+1} = (x_{i+1}, y_{i+1})$ and $y_i \leq y \leq y_{i+1}\}$. En la Figura 6.1 se ejemplifica esta definición.

Si representamos los slabs dentro de un arreglo ordenado por la mínima ordenada dentro de cada uno de ellos, es posible encontrar el slab al que pertenece (x, y) en tiempo $O(logN)$.

La intersección de un slab con las aristas de ψ define trapezoides. Como las aristas de ψ se intersectan sólo en sus vértices extremos (por ser un encaje planar) no existen intersecciones de aristas dentro del slab (ver Figura 6.2).

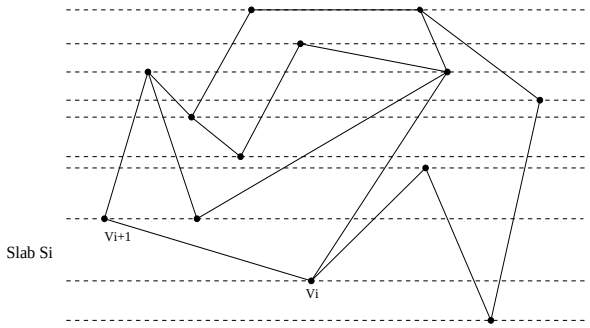


Figura 6.1: En línea contínua se muestra el encaje planar ψ y en línea punteada sus slabs.

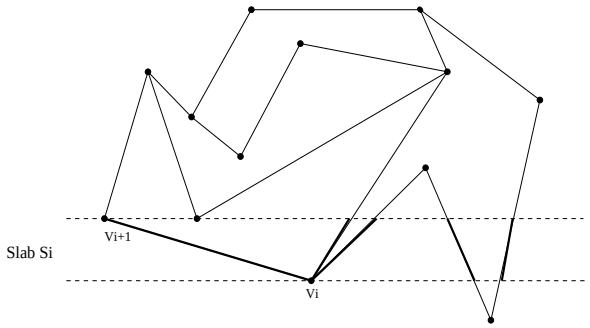


Figura 6.2: La aristas de ψ no se intersectan dentro de un slab.

Método	Espacio	Preprocesamiento	Consulta
Slab	$O(N^2)$	$O(N^2)$	$O(logN)$
Cadena	$O(N)$	$O(NlogN)$	$O(log^2N)$
Trapezoide	$O(NlogN)$	$O(NlogN)$	$O(4log_2N)$

Tabla 6.1: Métodos de localización de un punto en un encaje planar.

Si los elementos de U que cruzan un slab son ordenados de izquierda a derecha es posible utilizar el método de búsqueda binaria para determinar en tiempo $O(logN)$ el trapezoide que contiene a (x, y). Así, es posible determinar en tiempo óptimo la región de ψ que contiene el punto (x, y). De manera más precisa, cada slab se representa como un arreglo que contiene, los índices en el arreglo de nodos lado de las aristas de U que intersectan el slab, a partir de esta información es posible recuperar el nombre de la cara que contiene a (x, y) dentro de la $LADC(\psi)$.

Sólo resta analizar el espacio y tiempo de preprocesamiento [1] requerido por el método slab.

El espacio requerido es $O(N^2)$, ya que cada slab puede contener $O(N)$ segmentos de arista y hay $N-1$ slabs.

El tiempo de preprocesamiento debe reportar cada slab junto con las aristas que lo atraviesan debidamente ordenadas. Para ordenar de izquierda a derecha los segmentos de arista dentro de cada slab, se recurre a la *técnica de barrido en el plano*, en la que se recorren todos los slabs de abajo hacia arriba manteniendo el orden de las aristas de izquierda a derecha de la siguiente forma: cada vez que se encuentra un nuevo vértice, se insertan las aristas que inician en el vértice (pensando en que las aristas están dirigidas de abajo hacia arriba) y se borran las aristas que finalizan en el vértice en un árbol binario balanceado (estas operaciones son efectuadas en tiempo $O(logN)$) y entonces se reporta el slab con las aristas que lo atraviesan. Desde el punto de vista de la complejidad algorítmica, el trabajo consiste en borrar e insertar cada arista en un tiempo $O(logN)$ por operación y de la generación de la salida. Como un slab puede tener N aristas, el tiempo empleado para la inserción de las aristas es $O(NlogN)$ y para el borrado de las aristas es $O(NlogN)$, lo que da $O(NlogN)$. El tiempo empleado en generar la salida de un slab es $O(N)$ (ya que un slab puede contener N aristas), por lo que los $N-1$ slabs dan $O(N^2)$ en la generación de la salida. Así, el tiempo total de preprocesamiento es $O(NlogN) + O(N^2) = O(N^2)$.

A continuación se muestra el algoritmo. Los vértices son almacenados en el arreglo $VERTICE$, ordenados ascendentemente por la coordenada y; $B[i]$ es el conjunto de aristas que finalizan en el $VERTICE[i]$ (aristas entrantes), ordenadas en el sentido contrario de las manecillas del reloj; $A[i]$ es el conjunto de aristas que inician en el vértice $VERTICE[i]$ (aristas salientes), ordenadas en sentido contrario al de las manecillas del reloj. Las arista son mantenidas en el árbol balanceado L.

[1] Es el tiempo requerido para arreglar las estructuras de datos y poder realizar la búsqueda ([26])

Algorithm 6.1.1 *Preprocesamiento del algoritmo de localización de puntos.*

Entrada: *Encaje planar ψ con N vértices.*

Salida: *slabs cuyos segmentos que lo atraviesan están ordenados de izquierda a derecha.*

Método:
1. **begin** $VERTICE[1:2N]$:= *Conjunto V de ψ ordenados ascendentemente por y;*
2. $L \leftarrow \emptyset$;
3. **for** $i \leftarrow 1$ **until** N **do**
4. **begin** *BORRAR(B[i]);*
5. *INSERTAR(A[i]);*
6. *Reportar L;*
7. **end**
8. **end**

6.1.1 Observaciones al método slab

En la Tabla 6.1 se hace una comparación de las complejidades temporales y espaciales de diferentes métodos de localización de un punto en un encaje planar. Puede verse que el método slab excede a los métodos de la Cadena (ver [25]) y del Trapezoide (ver [9]) en el espacio requerido y tiempo de preprocesamiento; sin embargo, resulta ser más óptimo en la operación de consulta de un punto dado. Como el tiempo de consulta es fundamental para el buen funcionamiento de un SIG, se optó por instrumentar el método slab; que además, es relativamente sencillo de programar.

Sumario

En el capítulo se mostró el funcionamiento del método slab que permite encontrar la región de un encaje planar que contiene a un punto dado. Se encontró que este es uno de los métodos más eficientes para resolver el problema de la localización de puntos y de fácil implantación. En el disco compacto anexo al documento de tesis se muestra el código en lenguaje C del método slab.

Capítulo 7

Validación de algoritmos instrumentados

La instrumentación de los algoritmos aquí presentados es en extremo delicada por la gran cantidad de detalles y casos especiales que se pueden presentar; además, la dimensión de los volúmenes de información es tal que detectar los errores es una tarea complicada. Las pruebas efectuadas no sólo han permitido detectar errores de lógica de programación, sino que también errores más sutiles como los causados por las imprecisiones de los cálculos aritméticos efectuados por la computadora, tal como ocurrió en el algoritmo de Bentley/Ottmann (ver Capítulo 3). En la Tabla 7.1 se muestra el tipo de error que presentó cada uno de los programas desarrollados.

Con las pruebas utilizadas, se ha pretendido garantizar un alto nivel de confiabilidad en nuestros programas, pero no estamos dando un nivel específico porque para hacerlo es necesario recurrir a técnicas estadísticas que están fuera de los objetivos de nuestro trabajo (ver [36, 37]). Sabemos que los errores en un desarrollo se presentan como una superposición de variables aleatorias exponenciales; al iniciar el proceso de depuración, los errores son tantos que simplemente aparecen uno tras otro, cuando se ha logrado un cierto nivel de confiabilidad, cada corrección introduce nuevos errores pero una vez que se alcanza la estabilidad, los períodos de no fallo tienden a crecer exponencialmente en el tiempo. Nuestras pruebas se han corrido por tiempos tan largos como se ha podido y en estos tiempos no hemos detectado más errores, pero, como ya lo hemos dicho no damos una garantía de que se hayan eliminado todos.

Programa	**Errores lógicos**	**Errores de imprecisión aritmética**
Decodificador DXF	si	no
Bentley/Ottmann	si	si
Edelsbrunner	si	no
Construcción de la LADC	si	no
Servidor de consultas espaciales	si	no

Tabla 7.1: Errores que presentó cada uno de los programas desarrollados en la tesis.

Nuestras pruebas han sido:

1. Pruebas interactivas con el usuario. En este tipo de pruebas el usuario proporciona al algoritmo los datos de entrada (a través de un archivo o de la entrada estándar), y después de que los datos son procesados por el algoritmo, el usuario recibe los datos de salida (a través de un archivo o de la salida estándar) con los cuales puede verificar la correctitud de los resultados.

2. Pruebas aleatorias. Este tipo de pruebas genera de manera aleatoria los datos de entrada que alimentan a un algoritmo dado y se crean funciones que verifican la integridad de las estructuras de datos cada vez que un nuevo dato es alimentado al algoritmo.

7.1 Ejemplo de validación de algoritmos

El ejemplo que sigue muestra el proceso llevado a cabo para validar los algoritmos instrumentados en el presente trabajo basado en las pruebas de tipo interactivo. Aunque, en principio, el ejemplo puede parecer trivial (debido al número de segmentos de línea que contiene), resulta ser muy instructivo para mostrar el proceso de validación seguido. El lector puede encontrar, en el disco compacto anexo al presente documento, ejemplos mas robustos y utilizarlos para llevar a cabo sus propias puebas basándose en este ejemplo (ver apéndice A).

El ejemplo que sigue se basa en el archivo *malla.dxf*. En el inciso (a) de la Figura 7.1 se muestra el archivo original. Los puntos de mayor tamaño denotan los extremos de los segmentos rectilíneos (nótese que no están reportadas todas las intersecciones). En (b) se ubica el archivo dentro de un plano coordenado. Para encontrar todas las intersecciones del archivo en (a), tenemos que ejecutar la siguiente secuencia de comandos desde el "shell" de Linux (ver apéndice A para mas información acerca del proceso de instalación y ejecución de los programas):

$./dxf.out < ./mapas/malla.dxf \mid ./filtrodxf\ -e \mid ./interseg.out$

Donde:

$./dxf.out$	es el programa que decodifica el formato DXF.
$./mapas/malla.dxf$	es el subdirectorio que contiene el archivo original.
$./filtrodxf - e$	es el programa que filtra la información espacial.
$./interseg.out$	es el programa de intersecciones.

Nótese que en la ejecución de estos programas se hace uso de las tuberías [1] usadas en los sistemas UNIX; así por ejemplo, la salida del decodificador *DXF* alimenta la entrada del programa que filtra la información espacial, y la salida de éste último, alimenta la entrada del programa de intersecciones.

La salida de la secuencia de comandos anterior puede verse representada en el inciso (c) de la Figura 7.1, en la que han sido recortados todos los segmentos de línea, de tal manera, que cualquier segmento de línea solo tiene intersecciones, a lo mas, en sus extremos. La

[1] Pipes en inglés

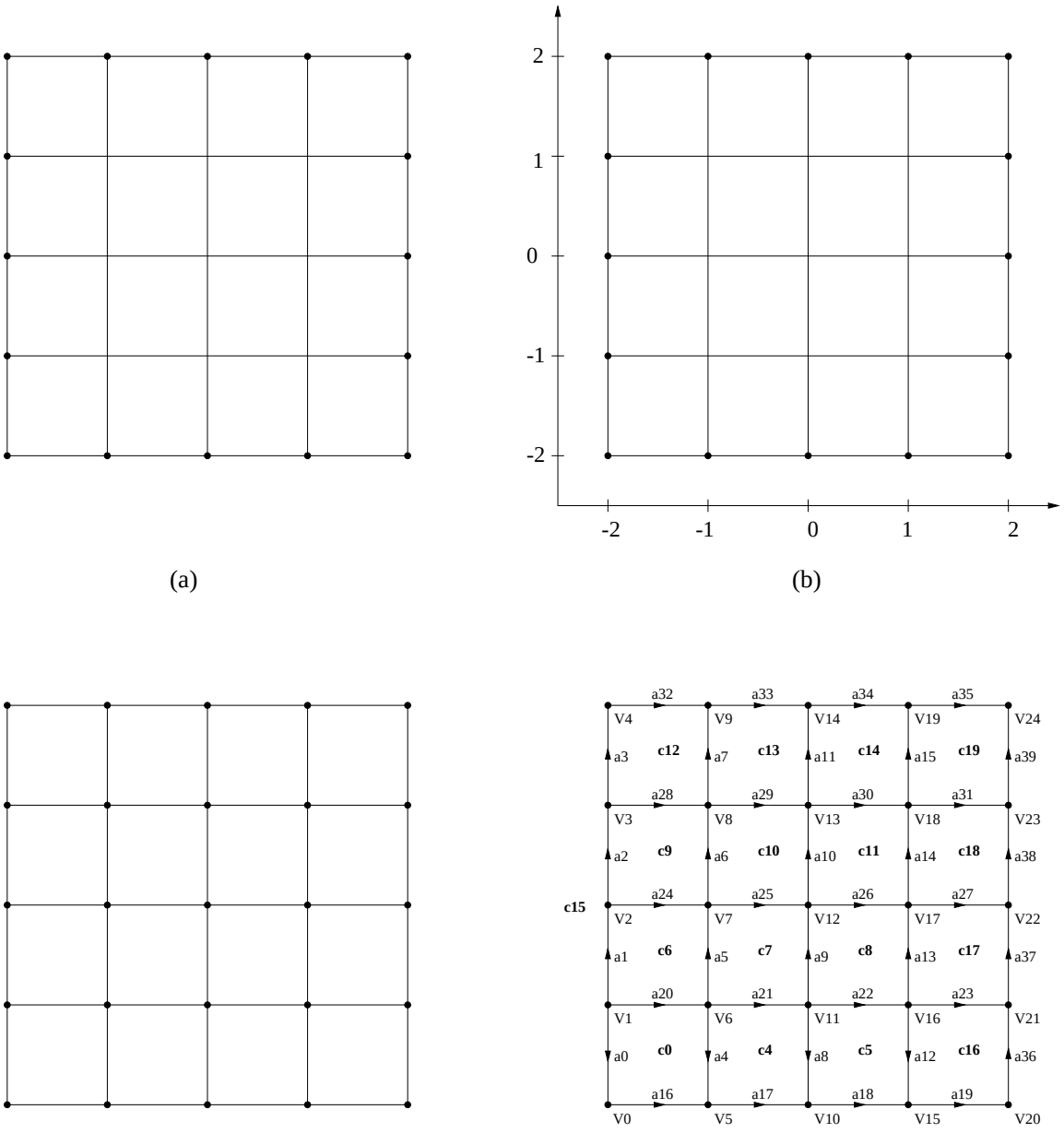


Figura 7.1: Archivo *malla.dxf*. En (a) se muestra el archivo original, en (b) se ubica el archivo dentro de un plano coordenado (nótese que no están reportadas todas las intersecciones), en (c) después de ser procesado por el programa de intersecciones y en (d) después de ser procesado por el programa que construye la LADC.

salida de la secuencia de comandos anterior puede ser enviada a un archivo en formato *scr* [2] que puede ser leido por AutoCAD y, de esta manera, verificar que el recorte se los segmentos de línea sea el correcto. Para hacer lo anterior, se debe ejecutar la siguiente secuencia de comandos:
./dxf.out < ./mapas/malla.dxf | ./filtrodxf −e | ./interseg.out | ./interseg_scr.out
Donde:
./interseg_scr.out Es el programa de conversión al formato *scr*.

Para construir la LADC del encaje planar del inciso (c) de la Figura 7.1, tenemos que ejecutar en el "shell" de Linux la siguiente secuencia de comandos:
./dxf.out < ./mapas/malla.dxf | ./filtrodxf −e | ./interseg.out | ./ladc.out
Donde:
./ladc.out es el programa que construye la LADC (ver Capítulo 5).

A continuación se muestra parte de la salida producida por la secuencia de comandos anterior:

NODOS_LADO:

a0: 1 0 0 15 1 16 0

a1: 1 2 15 6 20 2 0

a2: 2 3 15 9 24 3 0

a3: 3 4 15 12 28 32 0

a4: 6 5 4 0 20 17 0

a5: 6 7 6 7 21 24 0

a6: 7 8 9 10 25 28 0

a7: 8 9 12 13 29 32 0

a16: 0 5 0 15 0 4 0

a20: 1 6 6 0 0 5 0

a24: 2 7 9 6 1 6 0

a28: 3 8 12 9 2 7 0

a32: 4 9 15 12 3 33 0

VERTICES:

v0: 16, -2.000000, -2.000000,0.000000

v1: 20, -2.000000, -1.000000,0.000000

v2: 24, -2.000000, 0.000000,0.000000

v3: 28, -2.000000, 1.000000,0.000000

v4: 32, -2.000000, 2.000000,0.000000

v5: 17, -1.000000, -2.000000,0.000000

v6: 21, -1.000000, -1.000000,0.000000

v7: 25, -1.000000, 0.000000,0.000000

v8: 29, -1.000000, 1.000000,0.000000

v9: 33, -1.000000, 2.000000,0.000000

CARAS:

c0: 20

c6: 24

c9: 28

[2] Script de AutoCAD

```
c12: 32
c15: 39
```

La información anterior describe parte de la LADC (ver Capítulo 5) del encaje planar del inciso (c) de la Figura 7.1. La etiqueta NODOS_LADO índica que sigue la información de las aristas en el formato *#arista: vértice_inicial, vértice_final, cara_izquierda, cara_derecha, rotación1, rotación2*, la etiqueta VERTICES índica que sigue la información de los vértices en el formato *#vértice: arista_incidente_de_ mayor_grado, coordenada_x, coordenada_y, coordenada_z* y la etiqueta CARAS índica que sigue la información de las caras en el formato *#cara: arista_incidente_de_mayor_grado*. Esta información puede ser utilizada para construir parte de la LADC del inciso (d) de la Figura 7.1.

Con el programa *dceltst.out* podemos validar la estructura LADC de un mapa dado, ya que permite llevar a cabo operaciones de navegación sobre la LADC tal como fue descrito en la Sección 5.4. El uso del programa es el siguiente:
dceltst.out archivo_ladc
Donde:
archivo_ladc Es el archivo que contiene la estructura de datos LADC.

Cabe señalar que ésta no es la única forma de validar la LADC, ya que como se menciono anteriormente, se instrumentarón funciones que verifican de manera automática la integridad de dicha estructura.

Para ejecutar el servidor de consultas espaciales se debe emplear la secuencia de comandos:
$./dxf.out < ./mapas/malla.dxf \mid ./filtrodxf\ -e \mid ./interseg.out \mid ./ladc.out \mid ./sloc\_pto.out$
Donde:
./sloc_pto.out Es el programa servidor de consultas espaciales.

La secuencia de comandos previa crea en la memoria de la computadora los "slabs" (ver Capítulo 6) que permitiran localizar la región (cara de la LADC) que contiene a un punto dado.

Debido a que el servidor de consultas espaciales se basa en el modelo cliente-servidor podemos consultar la cara de la LADC que contiene a un punto dado de la siguiente manera:
$./cloc\_pto.out - p\ 0.5, 0.5, 0$
$./cloc\_pto.out - p\ 0.6, 0.6, 0$
$./cloc\_pto.out - p\ 0.1, 0.1, 0$
$./cloc\_pto.out - p\ 0.9, 0.9, 0$
Donde:
./cloc_pto.out Es el programa cliente del servidor de consultas espaciales.

Así, encontraremos que los puntos $(0.5, 0.5, 0), (0.6, 0.6, 0), (0.1, 0.1, 0)$ y $(0.9, 0.9, 0)$ se encuentran en la cara 11 de la LADC, tal como puede apreciarse en el inciso (d) de la Figura 7.1.

Conclusiones

En el presente trabajo de tesis se instrumentaron los módulos de lo que hemos denominado *núcleo de un Sistema de Información Geográfica* correspondientes a la representación, manipulación y consulta de la información espacial contenida en mapas.

Se instrumentó el módulo que resuelve el problema de las intersecciones de segmentos rectilíneos sobre el plano. Para esto, se programó el método de Bentley/Ottmann, que aunque correcto formalmente, es impráctico en entornos de programación de punto flotante.

El problema de la sensibilidad numérica fue resuelto con una modificación relativamente simple al método de Edelsbrunner que sirve para encontrar las intersecciones de rectángulos sobre el plano. Esta solución constituye una alternativa realmente interesante para resolver el problema de las intersecciones de segmentos rectilíneos sobre el plano.

Tanto el método de Bentley/Ottmann como el de Edelsbrunner pueden llegar a tener una complejidad temporal cuadrática en el peor caso; sin embargo, en el caso promedio su comportamiento temporal es $O(NlogN)$, lo que resulta ser óptimo.

El método de Bentley/Ottmann puede ser rescatado si se instrumenta una aritmética de punto flotante distinta, como puede ser: aritmética racional ([20]), aritmética con precisión infinita (como lo hacen matemática, Scheme y [15]), entre otras. No intentamos llevar a cabo la instrumentación de alguna de estas aritméticas ya que consideramos queda fuera de los objetivos primordiales de la tesis, sin embargo, se propone la instrumentación para ser considerada como trabajo futuro.

Se instrumentó el módulo LADC (Lista de Aristas Doblemente Conectadas) que consiste en una estructura de datos para representar mapas internamente en la memoria de la computadora y poder llevar a cabo consultas eficientes sobre los mismos. La LADC encuentra aplicaciones en el diseño de circuitos VLSI [22], la simplificación de redes por medio de reducciones $\Delta - Y$ [14], así como, en estudios relacionados con la Teoría Topológica de las Gráficas [17]. Análogamente, se crea un método de navegación en mapas basado en la idea de mapas combinatorios de Tutte [34].

Se instrumentó el módulo que encuentra la región de un mapa que contiene a un punto dado empleando el método slab cuyo comportamiento temporal de preprocesamiento es $O(N^2)$ y de consulta es $O(logN)$. Este módulo resulta fundamental para la realización de consultas en un SIG.

Finalmente, se instrumentó el módulo que lleva a cabo la decodificación del formato *DXF* de AutoCAD que permite extraer la información espacio-nominal de un mapa dado.

Como trabajo futuro queda la terminación del núcleo del Sistema de Información Geográfica propuesto. Básicamente se tienen que instrumentar los módulos correspondientes a la obtención y manipulación de la información nominal y el módulo encargado de la visualización y despliegues gráficos.

Apéndice A

Manual de usuario

A continuación se describe el contenido del disco compacto adjunto a la tesis, así como el proceso de instalación y ejecución de los programas desarrollados. También se muestra el prototipo de un desarrollo en Internet para la realización de búsquedas espacio-nominales en mapas.

A.1 Contenido del disco compacto

En el disco compacto adjunto se encuentra la siguiente información:

1. Directorio /. El directorio raíz contiene los archivos:

   (a) *ayuda.doc*

   (b) *instalador.sh*

   El primer archivo describe el contenido del disco compacto y explica cómo instalar los programas desarrollados en la tesis, el segundo archivo es el instalador de dichos programas.

2. Directorio */mapas*. Este directorio contiene mapas en formato *DXF*, utilizados para las demostraciones.

3. Directorio */documento*. Este directorio contiene propiamente el documento de tesis en formato *pdf*, *ps* y *dvi*.

4. Directorio */programas*. Este directorio contiene el archivo *programas.tar.gz* con todos los programas fuentes desarrollados en la tesis.

5. Subdirectorio */programas/fuentes*. Aquí se encuentran los siguientes subdirectorios:

   (a) */programas/fuentes/dxf*. Este subdirectorio contiene el código fuente del módulo decodificador del formato *DXF*.

   (b) */programas/fuentes/interseg*. Este subdirectorio contiene el código fuente del módulo de intersección de segmentos de línea.

(c) */programas/fuentes/ladc*. Este subdirectorio contiene el código fuente del módulo de construcción de la LADC.

(d) */programas/fuentes/loc_pto*. Este subdirectorio contiene el código fuente del servidor de consultas espaciales.

6. Subdirectorio */programas/pg*. Este subdirectorio contiene la biblioteca de programación generalizada de estructuras y algoritmos [31]. En particular, los programas desarrollados en la tesis hacen uso de las bibliotecas de árboles binarios, árboles balanceados, listas doblemente ligadas, de ordenamiento por el método del montículo, etc.

A.2 Instalación de los programas

Para instalar los programas desarrollados en la tesis, se deben copiar los archivos */programas/programas.tar.gz* e */instalador.sh* en el directorio $HOME del usuario. Posteriormente, desde el directorio $HOME del usuario se ejecuta el archivo *./instalador.sh*, mismo que se encarga de descomprimir y compilar los programas generándose en el directorio *$HOME/progs* los siguientes archivos ejecutables:

1. *dxf.out*. Programa que decodifica el formato *DXF*.

2. *filtrodxf*. Programa que filtra la salida del decodificador del formato *DXF*. El programa permite filtrar tanto la información espacial como nominal.

3. *interseg.out*. Programa que encuentra todos los pares de intersecciones de un conjunto dado de segmentos rectilíneos.

4. *ladc.out*. Programa que construye la estructura de datos LADC.

5. *cloc_pto.out*. Programa cliente del servidor de consultas espaciales.

6. *sloc_pto.out*. Programa servidor de consultas espaciales.

A.3 Ejemplos de ejecución de los programas

Los programas generados en la sección anterior, permiten ejecutar programas siguiendo la filosofía de los sistemas UNIX sobre el uso de tuberías [1]. Así, es posible que la salida del decodificador *DXF* alimente la entrada del programa de intersecciones, o bien, que la salida de éste último alimente la entrada del programa que construye la LADC. Con los ejemplos que siguen, se muestra cómo puede usarse la salida de un programa para alimentar la entrada de otro.
La ejecución de estos ejemplos debe hacerse desde el subdirectorio *$HOME/progs*.

[1] Pipes en inglés

1. Para decodificar el mapa *copalita.dxf* se debe ejecutar la siguiente línea:
   ./dxf.out -l < Tdiv_pol ./mapas/copalita.dxf
   Donde:
   ./dxf.out es el archivo ejecutable del decodificador.
   -l especifica las capas, separadas por comas, a decodificarse.
   Tdiv_pol es el nombre de la capa.

2. Para filtrar la información espacial del mapa *copalita.dxf* se debe ejecutar la siguiente línea:
   ./dxf.out -l < Tdiv_pol ./mapas/copalita.dxf | ./filtrodxf -e
   Donde:
   ./filtrodxf es el archivo ejecutable para filtrar la información espacial y/o nominal.
   -e filtra la información espacial (*-n* filtra la información nominal).

3. Para encontrar las intersecciones del mapa *copalita.dxf*:
   ./dxf.out -l < Tdiv_pol ./mapas/copalita.dxf | ./filtrodxf -e | ./interseg.out
   Donde:
   ./interseg.out es el archivo ejecutable para encontrar las intersecciones.

4. Para construir la LADC del mapa *copalita.dxf*:
   ./dxf.out -l < Tdiv_pol ./mapas/copalita.dxf | ./filtrodxf -e | ./interseg.out | ./insa.out
   Donde:
   ./insa.out es el archivo ejecutable que construye la LADC.

5. Para encontrar la región que contiene el punto $0.63, 0.21, 0$ en el mapa *copalita.dxf*:
   ./dxf.out < ./mapas/planobase.dxf | ./filtrodxf -e | ./interseg.out | ./insa.out | ./sloc_pto.out&
   ./cloc_pto.out -p 0.63,0.21,0

A.4 Prototipo en Internet para la consulta de información espacio-nominal de un mapa dado

En la tesis se ha propuesto el diseño del Núcleo de un Sistema de Información Geográfica (ver Capítulo 1) y se han desarrollado los algoritmos básicos para llevar a cabo consultas de tipo espacial en un mapa dado.

Paralelamente a este desarrollo, se ha llevado a cabo la implementación de dos prototipos en Internet para la consulta de información espacio-nominal en un mapa. Dichas implemantaciones han sido desarrolladas por mi asesor de tesis y puede verse su funcionamiento en las URL dadas en [1] y [2]. Los dos prototipos permiten llevar a cabo consultas espaciales, pero solo el segundo realiza consultas nominales.

La importancia de los prototipos es que integran los algoritmos desarrollados en la tesis para llevar a cabo consultas de tipo espacial. Así, tenemos que el decodificador del formato *DXF* extrae la información tanto espacial como nominal de un mapa; el algoritmo de intersección de segmentos de línea recorta los segmentos para que las regiones de un mapa queden

perfectamente definidas; el algoritmo que construye la LADC sirve para definir las regiones de un mapa, y por último, el modelo cliente-servidor del Servidor de Consultas Espaciales permitirá, en un futuro, asociar la información nominal a una determinada región del mapa.

Bibliografía

[1] *http://148.247.14.228/~mathdoc/cna02.html.*

[2] *http://148.247.14.228/~sanjuan/fsanjuan.html.*

[3] SQL/86, Database Language SQL. Tech. Rep. ANSI X3, 135, American National Standards Institute, 1986.

[4] Autodesk Inc. *AutoCAD Release 12 Customization Manual*, May 1992.

[5] Bentley, J. L., and Ottmann, T. A. Algorithms for Reporting and Counting Geometric Intersections. *IEEE Trans. Comput. c-28* (1979), 643–647.

[6] Bollobás, B. *Extremal Graph Theory.* Academic Press, London-New York-San Francisco, 1978.

[7] Bondy, J. A., and Murty, U. *Graph Theory with Applications.* Macmillan Press, London and Basingstroke, 1976.

[8] Borrough, P. A. *Principles of Geographical Information Systems for Land Resource Assessment.* Clarendon Press, EE.UU., Oxford, Reino Unido, 1986.

[9] Chiang, Y.-J., and Tamassia, R. Dynamization of the trapezoid method for planar point location in monotone subdivisions. *J. Comput. Geom. Appl. 2* (1992), 311–333.

[10] Cohn, D. S. *AutoCAD.* Addison-Wesley, EE.UU., 1988.

[11] Demers, M. *Fundamentals of Geographic Informations Systems.* John Wiley & Sons, EE.UU., 1997.

[12] Dueker, J. K. y. D. K. Multipurpose Cadastre: Terms and Definitions. *VA: American Society for Photogrammetry and Remote Sensing y ACSM* (1989).

[13] Dybvig, R. K. *The Scheme Programming Language.* Prentice-Hall, 2nd edition, New Jersey, 1996.

[14] Feo, T. A., and Provan, J. S. Delta-Wye transformations and the efficient reduction of two-terminal planar graphs. *Operations Research 41-3* (may-june 1993).

[15] Fortune, S., and Wyk, C. V. Efficient exact arithmetic for computational geometry. *In Proceedings of the 9th ACM Symposium on Computational Geometry* (may 1993), 163–172.

[16] Goodman, J. E., and O'Rourke, J. *Handbook of Discrete and Computational Geometry.* CRC Press, 1997.

[17] Gross, J. L., and Tucker, T. W. *Topological Graph Theory.* John Wiley & Sons, EE.UU., 1987.

[18] Hopcroft, J. E., and Ullman, J. D. *Introducción a la Teoría de Autómatas, lenguajes y computación.* CECSA, México, 1995.

[19] Hutchinson, S., and Daniel, L. *Inside ArcView GIS.* Paperback, 3rd edition, Dec. 1999.

[20] Jaillon, P. 'LEA', a lazy exact arithmetic: Implementation and related problems. *Technical Report in preparation, École Nationale Supérieure des Mines de Saint-Étienne* (1993).

[21] Korth, H. F., and Silberschatz, A. *Fundamentos de bases de datos.* McGraw Hill.

[22] Kung, S. Y. *VLSI Array Processoors.* Prentice Hal, Prentice Hall, 1987.

[23] Momjian, B. *PostgreSQL Introduction and Concepts.* Addison-Wesley, First Printing, Nov. 2000.

[24] Omura, G. *Auto-CAD 12.* Gustavo Gili, Barcelona, 1994.

[25] Preparata, F. P., and Shamos, M. I. *The chain method.* In [26], 1985, pp. 48–60.

[26] Preparata, F. P., and Shamos, M. I. *Computational Geometry - An Introduction.* Springer-Verlag, New York, 1985.

[27] Preparata, F. P., and Shamos, M. I. *The doubly-connected-edge-list(DCEL).* In [26], 1985, pp. 15–17.

[28] Preparata, F. P., and Shamos, M. I. *Intersections of Rectangles and Related Problems.* In [26], 1985, pp. 351–355.

[29] Preparata, F. P., and Shamos, M. I. *The slab method.* In [26], 1985, pp. 45–48.

[30] Roberson, N., and Vitray, R. *Representativity of surface embeddings, Paths, flows and VLSI-Layout.* Springer Verlag, Berlin, 1990.

[31] Sagols, F. D. Programación Generalizada de Estructuras y Algoritmos. Tech. Rep. 13750/91, CINVESTAV, México, D.F., 1991.

[32] Sagols, F. D. *Hamiltonian Representation of Vox-solids.* PhD thesis, CINVESTAV, México, D.F., Sept. 1997. Cátedra Patrimonial CONACyT:940184.

[33] Stones, R., and Matthew, N. *Beginning Linux Programming.* Wrox, 2nd edition, 1985.

[34] Tutte, W. A theorem on planar graphs. *Trans AMS 82* (1956), 99–116.

[35] Vitray, R. *Representativity and flexibility of drawings of graphs.* PhD thesis, Ohio State University, 1987.

[36] Viveros, R., and Balakrishnan, N. Statistical Interference from Start-Up Demostration Test Data. *Journal of Quality Technology 25*, 2 (abr 1993), 119–130.

[37] Viveros, R., and Balakrishnan, N. Interval Estimation of Parameters of Life From Progressively Censored Data. *Technometrics 36*, 1 (feb 1994), 84–91.