Estructuras de datos geométricas

Winged-edge y Half-edge

Dr. Eduardo A. RODRÍGUEZ TELLO

CINVESTAV-Tamaulipas

12 de febrero del 2013

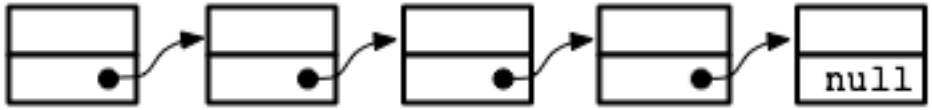
Cinvestav

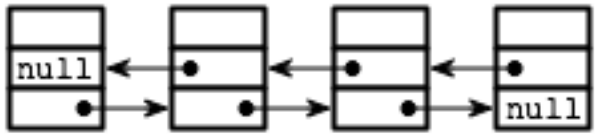
1. Estructuras de datos geométricas
   - Introducción
   - Winged-edge
   - Half-edge

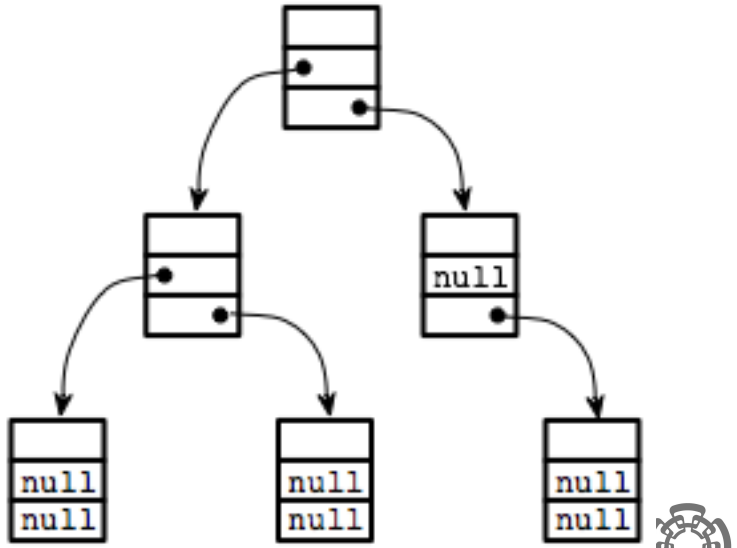
Cinvestav

Introducción

- Una estructura de datos es un repositorio de información
- La meta es organizar los datos de forma tal que:
  - El espacio de almacenamiento empleado sea mínimo
  - La recuperación de información (*query*) pueda procesarse rápidamente



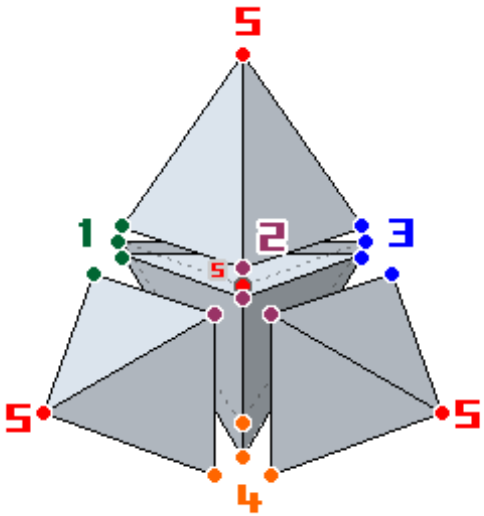


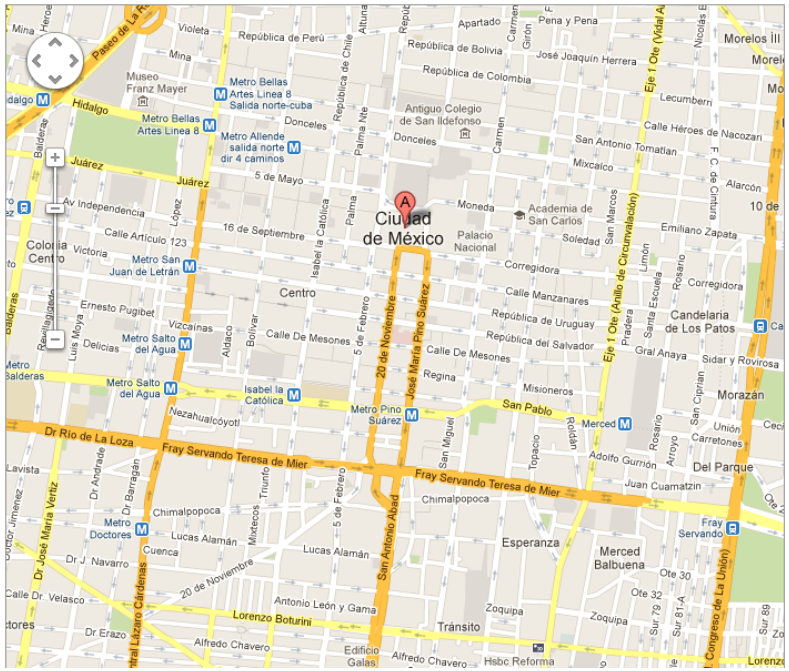


Cinvestav

Introducción

- Una estructura de datos geométrica (EDG) permite realizar una manipulación eficiente de la información topológica asociada a politopos en 2 o más dimensiones (vértices, aristas, caras)
- Las EDG se han convertido en una parte muy importante en nuestra vida cotidiana





Cinvestav

Introducción

- Algunos ejemplos incluyen las EDG conocidas como:
  - winged-edge
  - quad-edge
  - half-edge
  - quadtree
  - *Kd*-tree
  - BSP tree
  - Range tree

Cinvestav

1 Estructuras de datos geométricas
- Introducción
- Winged-edge
- Half-edge

Cinvestav

Winged-edge

Origen

- Fue propuesta por Baumgart en 1972, originalmente pensada para visión computacional

Características

- Usa principalmente las aristas (*edges*) para guardar la información importante del politopo
- Esta información es almacenada en una tabla, comúnmente llamada **tabla de aristas** (*edge table*)

• Bruce G. Baumgart. 1972. Winged Edge Polyhedron Representation. Technical Report. Stanford University, Stanford, CA, USA.
• Bruce G. Baumgart. 1975. A polyhedron representation for computer vision. In Proceedings of the National Computer Conference and Exposition (AFIPS 1975). ACM, New York, NY, USA, 589-596.

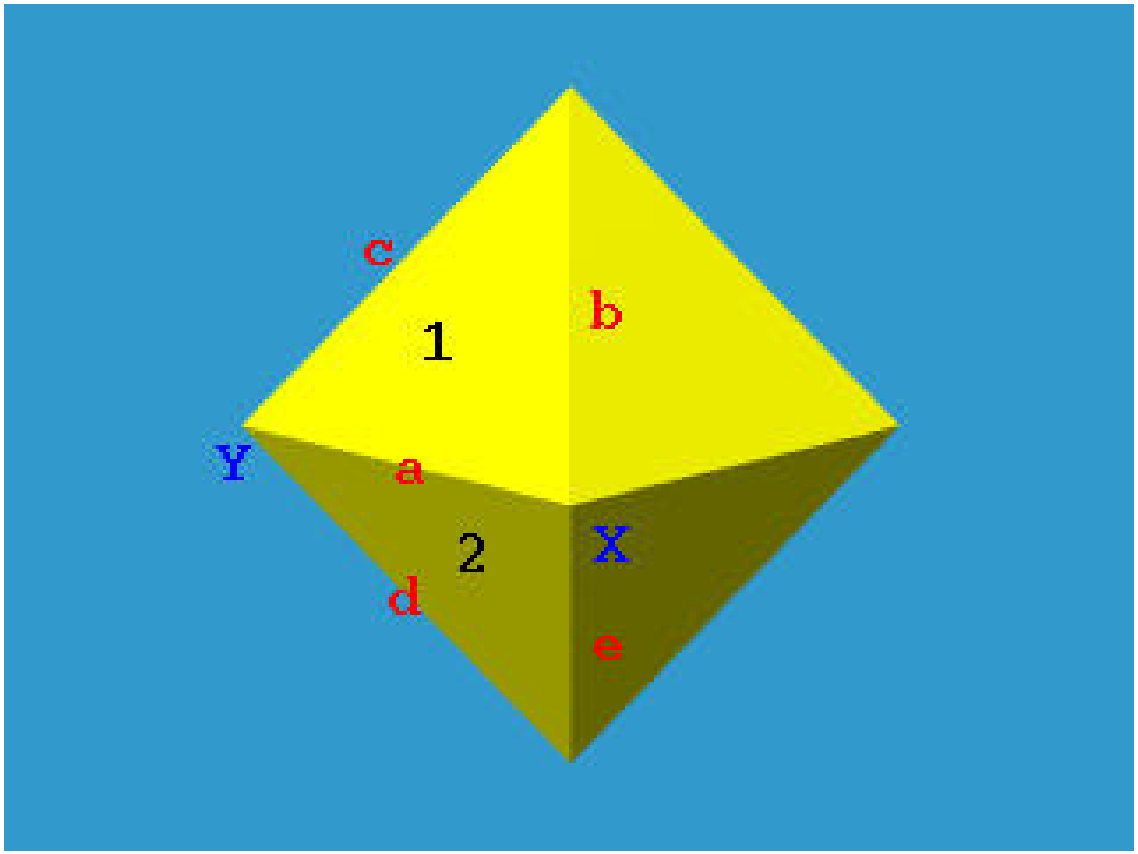
Cinvestav

Winged-edge

- Para construir la tabla de aristas necesitamos la siguiente información:
  1. Vértices de cada arista
  2. Caras izquierda y derecha de cada arista
  3. Arista predecesora y sucesora de cada arista al recorrer su cara izquierda
  4. Arista predecesora y sucesora de cada arista al recorrer su cara derecha
- Explicaremos estos puntos a continuación

Cinvestav

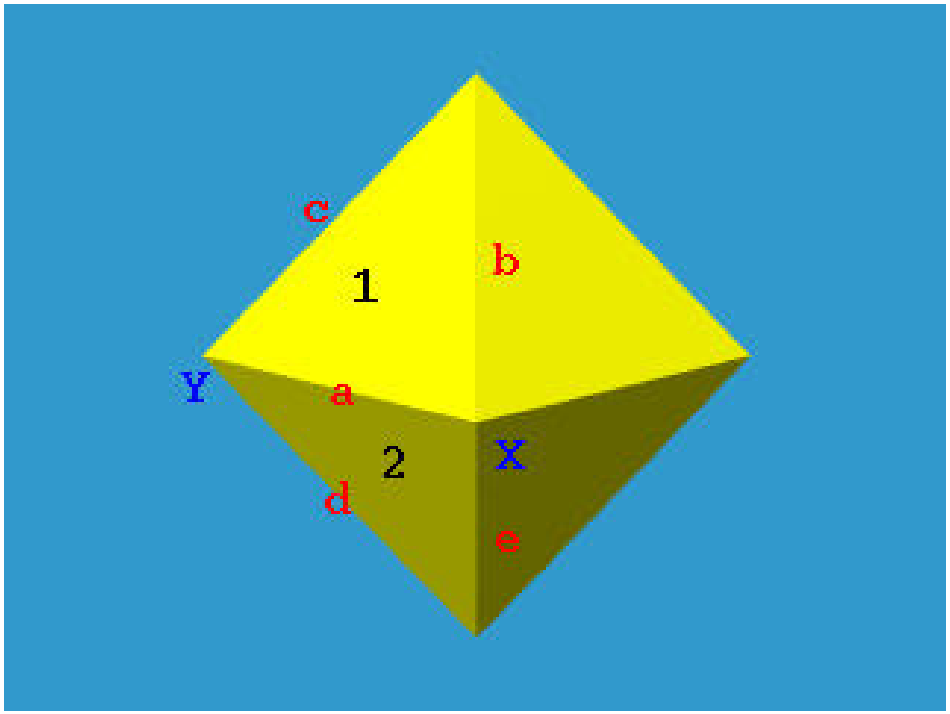
Winged-edge, notación



- Números corresponden a las **caras**
- Mayúsculas corresponden a los vértices
- Minúsculas corresponden a las aristas

Cinvestav

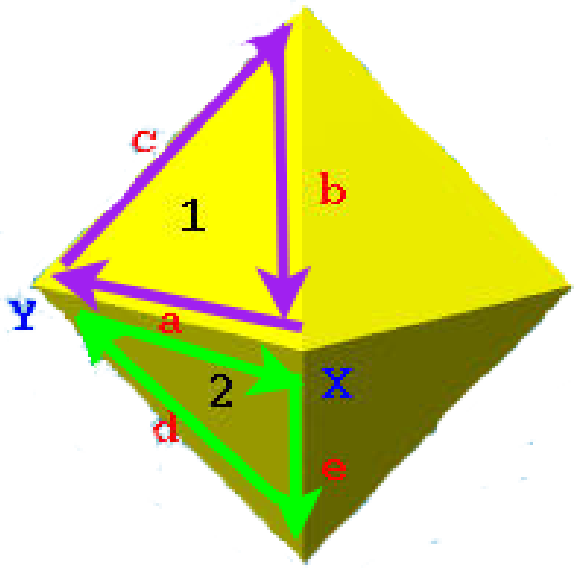
Winged-edge, vértices



- Concentrémonos en la arista a
- Vértices incidentes: Y y X
- Caras incidentes: **1** y **2**
- La cara **1** está delimitada por las aristas a, c y b
- La cara **2** está delimitada por las aristas a, e y d

Cinvestav

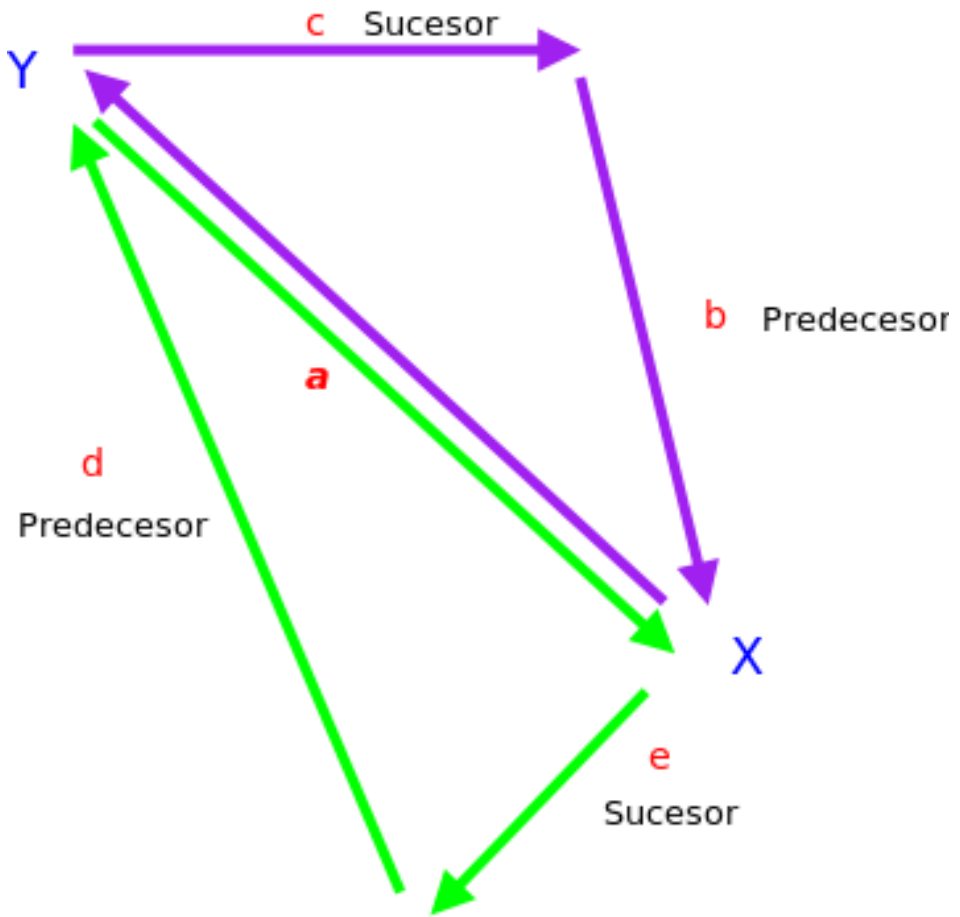
Winged-edge, caras izquierda y derecha



- Usando de referencia el sentido de las manecillas del reloj
- Si la dirección de la arista a es de X a Y
- La cara **1** corresponde a la cara derecha
- La cara **2** corresponde a la cara izquierda

Cinvestav

Winged-edge, aristas predecesoras y sucesoras



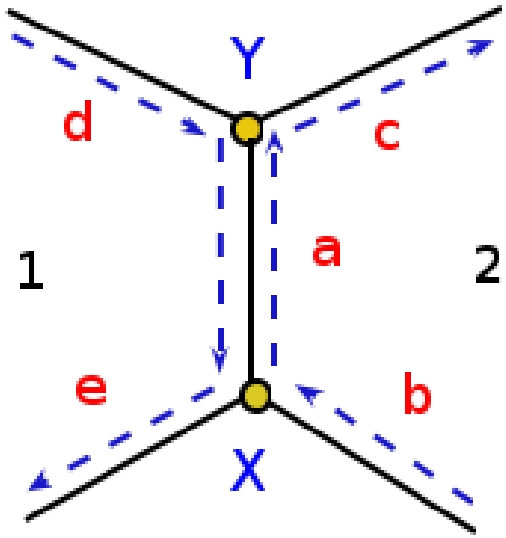
- Cada arista tiene una arista predecesora y otra sucesora
- Al recorrer las aristas de la cara **1** el predecesor y sucesor de la arista a son b y c
- Al recorrer las aristas de la cara **2** el predecesor y sucesor de la arista a son d y e

Cinvestav

Winged-edge, tabla de aristas

- Cada entrada de la tabla contiene la información de una arista, e.g., para la arista a

Arista	Vértices		Caras		Recorr. izq		Recorr. der	
	Inicio	Final	Der	Izq	Pred	Suc	Pred	Suc
a	X	Y	1	2	d	e	b	c



- Las cuatro aristas d, e, b y c son las alas de la arista a (*wings* → *winged-edge*)

Cinvestav

Winged-edge, tablas de vértices y caras

- La estructura de datos *winged-edge* requiere dos tablas adicionales:
  - Tabla de vértices
  - Tabla de caras
- La *tabla de vértices* tiene una entrada por cada vértice del politopo y contiene una arista que sea incidente a ese vértice.
- La *tabla de caras* tiene una entrada por cada cara y contiene una arista que sea parte de los límites de la cara

Cinvestav

Winged-edge, ejemplo

Tabla de aristas

Aristas	Vértices		Caras		Recorr. izq		Recorr. der	
	Inicio	Final	Der	Izq	Pred	Suc	Pred	Suc
a	A	D	2	1	e	f	b	c
b	A	B	1	4	c	a	f	d
c	B	D	1	2	a	b	d	e
d	B	C	2	4	e	c	b	f
e	C	D	2	3	c	d	f	a
f	A	C	4	3	d	b	a	e

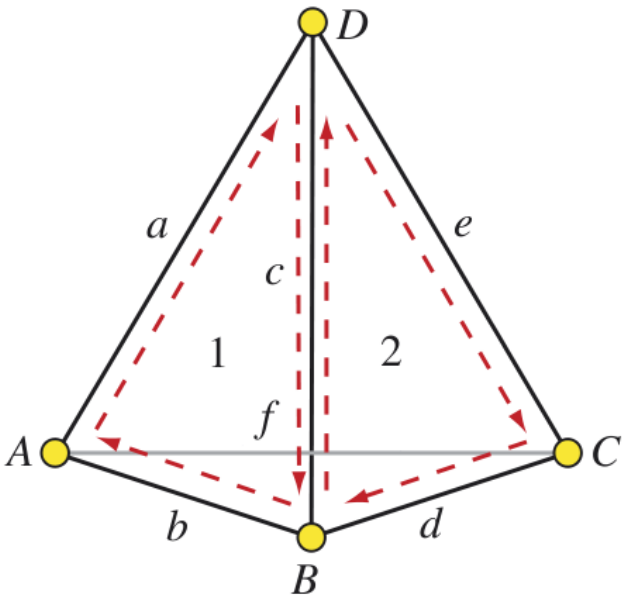


Tabla de vértices

Vértices	Arista
A	a
B	b
C	d
D	e

Tabla de caras

Caras	Arista
1	a
2	c
3	a
4	b

Cinvestav

Winged-edge, ventajas

- Debido a su organización *winged-edge* permite hacer un eficiente recorrido entre caras, aristas y vértices para contestar preguntas (*queries*) acerca de la adyacencia del politopo (existen 9 relaciones de adyacencia)
- Por ejemplo: ¿El vértice D es adyacente a la cara 3?, ¿Las caras 1 y 2 son adyacentes?
- La estructura de datos winged-edge puede contestar estas preguntas de adyacencia en forma muy eficiente e incluso algunas de ellas en tiempo constante

Cinvestav

Winged-edge, ventajas

- Por ejemplo, para encontrar todas las aristas incidentes a un vértice, simplemente se busca la arista incidente que se encuentra en la *tabla de vértices* y se siguen las aristas predecesoras o sucesoras a la siguiente arista
- Por eso, si una aplicación requiere de contestar muchas preguntas de topología del poliedro, *winged-edge* es una estructura más eficiente que las convencionales
- Funciona bien para representar caras con polígonos arbitrarios

Cinvestav

Winged-edge, ventajas

- El tamaño de las tablas se mantiene fijo una vez que se sabe cuantas caras, vértices y aristas componen al politopo
- *Winged-edge* es ideal para aplicaciones que necesiten geometría dinámica, como el modelado interactivo u otros donde haya cambios locales
- Recorrer el poliedro o detectar colisiones puede ser realizado de forma muy eficiente

Cinvestav

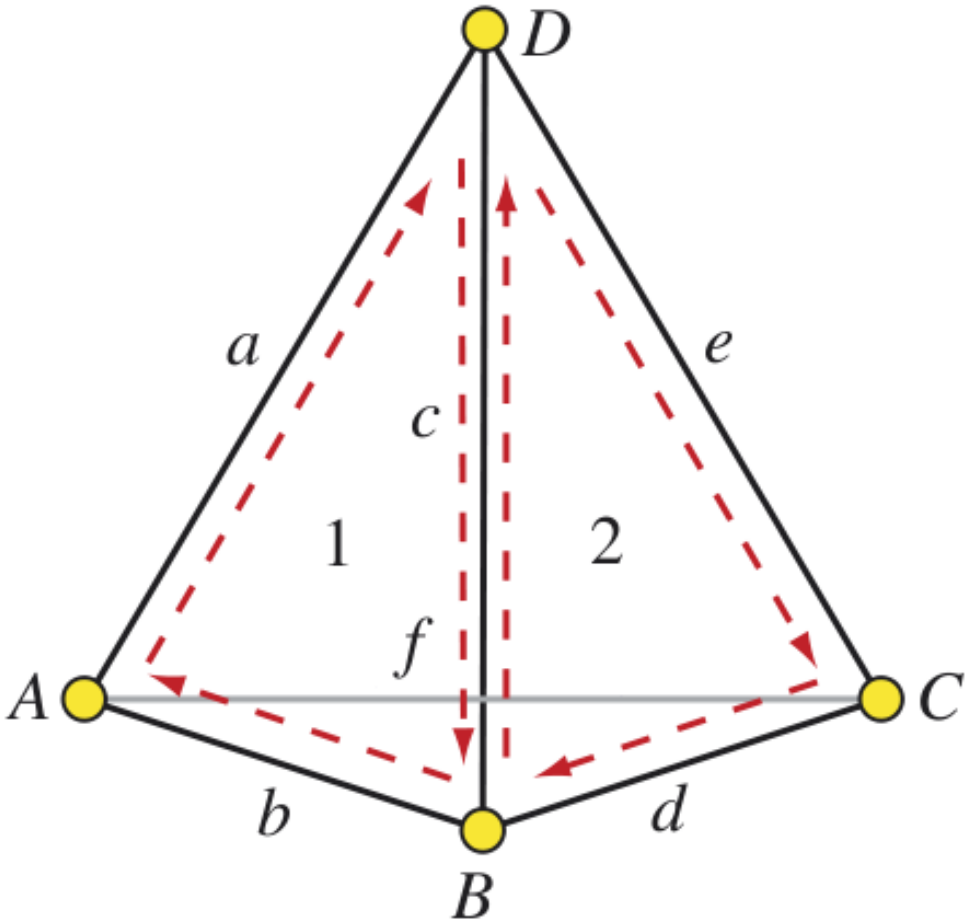
Winged-edge, un algoritmo de ejemplo

Encontrar todas las arista adyacentes de un vértice dado

```
1 algorithm vertexAdjacentEdge(v : Vertex);
2   structuretype WingedEdge
3   description "Mark the adjacent edges of a vertex."
4   var e, se : Edge;
5 begin
6   e := v.incidentEdge;
7   se := e;
8   do
9      mark e;
10     if e.startVertex = v then
11        e := e.leftSuccessor;
12     else
13        e := e.rightSuccessor;
14     end;
15  while e <> se;
16 end;
```

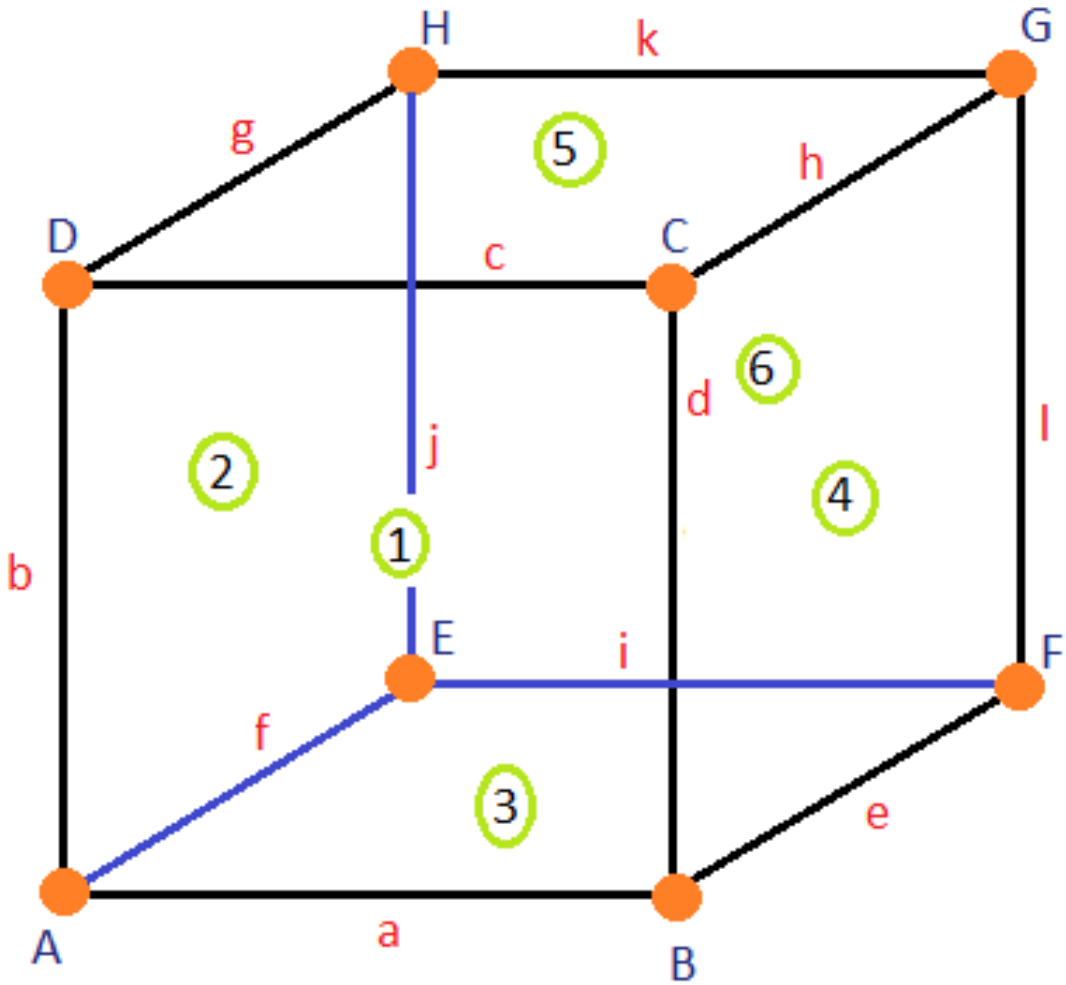
Winged-edge, programa de ejemplo

Tetraedro



Winged-edge, programa de ejemplo

Hexaedro



1 Estructuras de datos geométricas
- Introducción
- Winged-edge
- Half-edge

Cinvestav

Half-edge

Origen

- Fue propuesta de manera independiente por Muller y Preparata en 1978, y posteriormente por Eastman y Weiss en 1982

Características

- *Half-edge* es una estructura muy utilizada en muchos algoritmos de geometría computacional para manejar las subdivisiones poligonales del plano
- El principio que emplea es dividir cada arista en dos aristas dirigidas (*half-edges*)

• D. E. Muller and F. P. Preparata. Finding the intersection of two convex polyhedra. Theoretical Computer Science, Volume 7, Issue 2, 1978, Pages 217–236.
• C. M. Eastman, S.F. Weiss. Tree structures for high dimensionality nearest neighbor searching. Information Systems, Volume 7, Issue 2, 1982, Pages 115–122.

Cinvestav

Half-edge, estructura

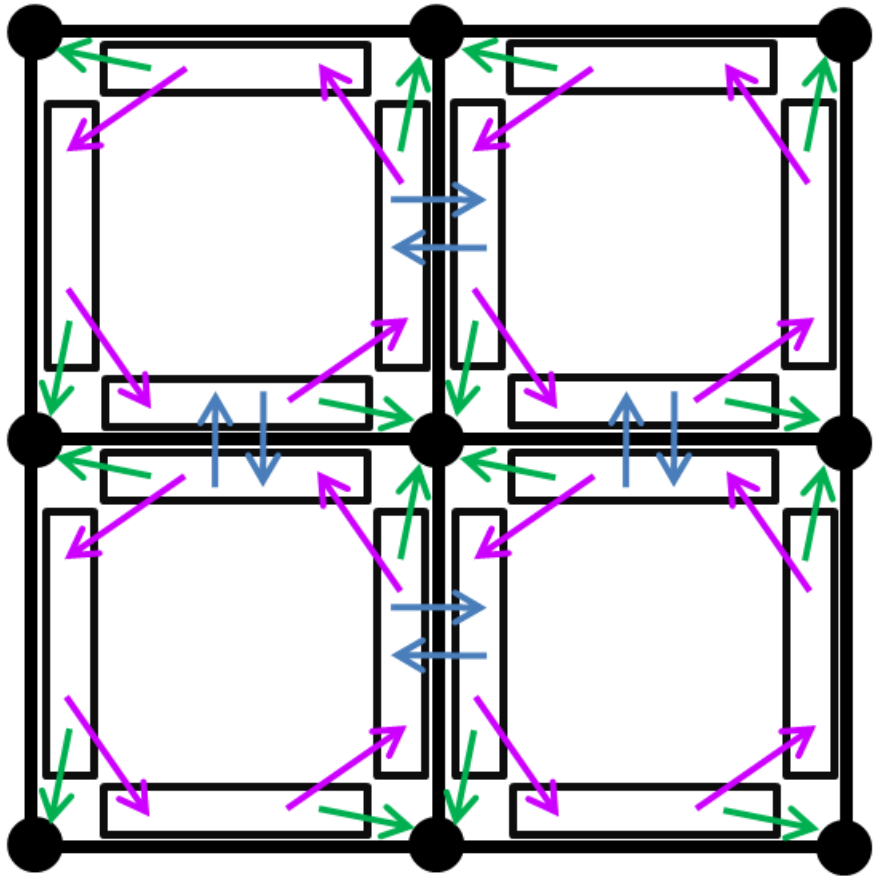
- Cada *half-edge* almacena la mitad de una arista
- A las dos mitades de una arista se les conoce como pares de una arista
- Las dos medias aristas son dirigidas y tienen direcciones opuestas

Cinvestav

Half-edge, estructura

Cada *half-edge* almacena la siguiente información:

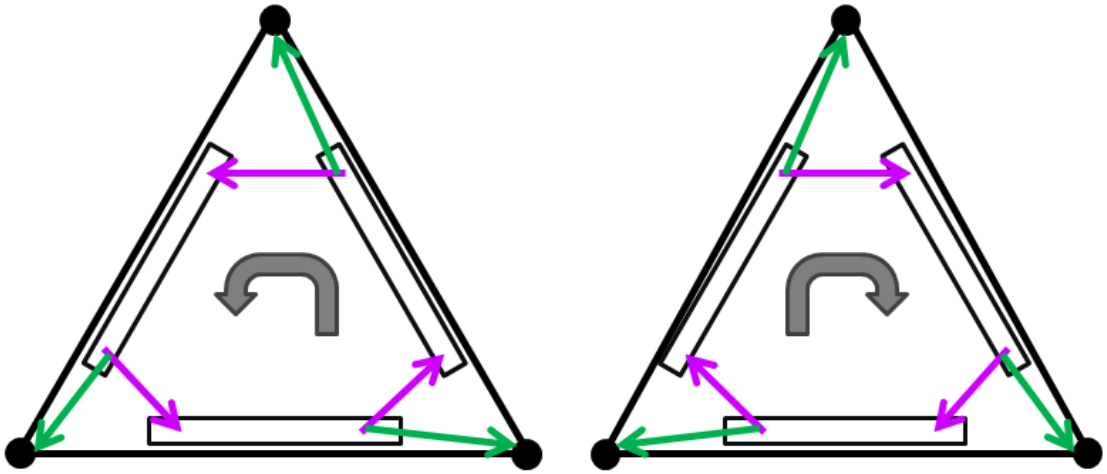
- Vértices
- *Half-edge* simétrico
- Siguiente *half-edge* de la cara izquierda (en sentido ↺)
- **Cara** (izquierda del *half-edge*)



Cinvestav

Half-edge, estructura

- Todos los *half-edges* que son frontera de una cara forman una lista circular ligada alrededor de su perímetro
- Las listas pueden ser orientadas en sentido de las manecillas del reloj o inverso, pero la orientación debe ser consistente en toda la representación



Cinvestav

Half-edge, estructura aristas

- Contiene:
  - Un apuntador al vértice adyacente al *half-edge*
  - Un apuntador a su *half-edge* par
  - Un apuntador a la cara de la cual el *half-edge* es frontera
  - Un apuntador al próximo *half-edge* dentro de la misma cara

```
struct HE-edge {
    HE-vertex* vert;
    HE-edge* pair;
    HE-face* cara;
    HE-edge* next;
}
```

Cinvestav

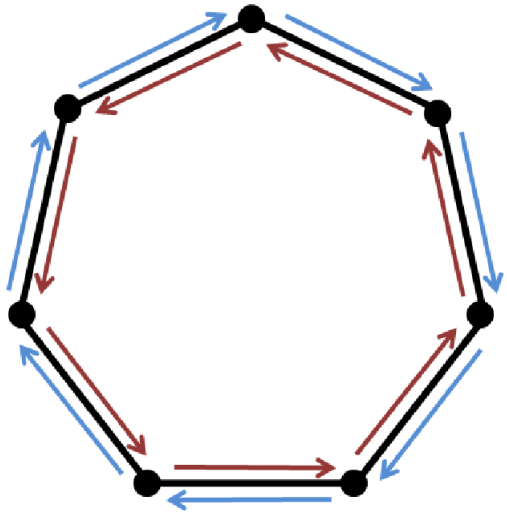
Half-edge, estructura vértices

- Contiene:
  - Las coordenadas del vértice
  - Un apuntador a un *half-edge* para el cual el vértice es el origen

```
struct HE-vertex {
    float x;
    float y;
    float z;
    HE-edge* edge;
}
```

Half-edge, estructura caras

- Contiene:
  - Un apuntador a un *half-edge* en su frontera (nulo para facetas sin fronteras)



```
struct HE-face {
    HE-edge* edge;
}
```

Half-edge, ejemplo de consultas

- Las respuestas a la mayoría de las consultas (*queries*) de adyacencia están almacenadas directamente en la estructura de datos
- Por ejemplo, para conocer las caras que son adyacentes a una arista (*half-edge*) pueden ser fácilmente encontradas con el siguiente código:

```
/* edge fue declarado de tipo HE-edge */
HE-vertex *vertex1 = edge→vert;
HE-vertex *vertex2 = edge→pair→vert;
```

Cinvestav

Half-edge, ejemplo de consultas

- Para conocer las caras que son adyacentes a una arista (*half-edge*) es suficiente utilizar el siguiente código:

```
/* edge fue declarado de tipo HE-edge */
HE-face *face1 = edge→cara;
HE-face *face2 = edge→pair→cara;
```

- Las dos consultas de adyacencia presentadas anteriormente pueden realizarse en tiempo constante ($O(1)$)

Cinvestav

Half-edge, ventajas

- A pesar almacenar información de adyacencia entre caras, vértices y aristas, el tamaño se mantiene constante (no es necesario utilizar arreglos dinámicos), además es razonablemente compacta
- Su complejidad temporal es lineal con respecto a la cantidad de información manejada
- La lista de adyacencias puede ser construida en tiempo proporcional al tamaño

Cinvestav