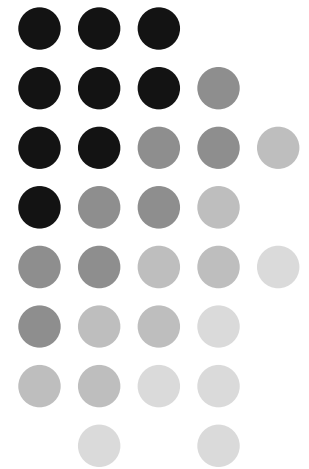


# Algoritmos sobre polígonos

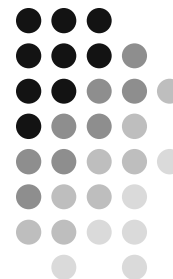


UNIVERSIDAD  
DE LOS ANDES

Diseño y Análisis de Algoritmos  
Cátedra de Programación  
Carrera de Ingeniería de Sistemas  
Prof. Isabel Besembel Carrera

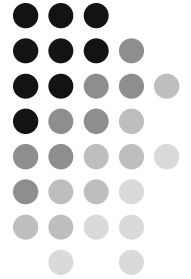


Parte del contenido está tomado de J. O'Rourke "Computational Geometry in C"

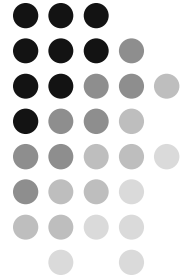


- Aplicaciones de ingeniería:
  - ❖ Representación de las propiedades espaciales y de las interrelaciones entre los dominios de las entidades.
  - ❖ En los sistemas de transporte, la planificación de rutas puede ser modelada utilizando un digrafo cuyos nodos representan entidades (e.g. aeropuertos, estaciones de metro e intersecciones de carreteras) y cuyas aristas capturan las conexiones entre los nodos
  - ❖ Sistemas de Información Geográfica
  - ❖ Diseño asistido por computador

# Datos espaciales

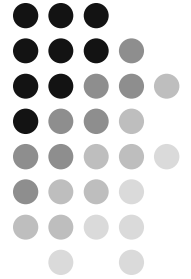


- Término para describir los datos que pertenecen al espacio ocupado por los objetos
  - ❖ Pueden ser geométricos y variados.
  - ❖ Se aplica a cualquier dato de un fenómeno distribuido en 2-, 3- o n-dimensiones que son ortogonales y homogéneas (D. Peuquet)
  - ❖ Consisten del conjunto de objetos espaciales construidos con: puntos, líneas, regiones, rectángulos, superficies, volúmenes, etc.
    - Ejm: Ciudades, ríos, estados, etc



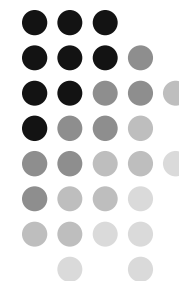
- Puede ser:
  - ❖ discreto o continuo,
  - ❖ limitado o no
  - ❖ absoluto o relativo
- Si se considera limitado normalmente se aplica un rango de  $70 \times 10^6$  mts.
- Ejm:  
 $52.62^\circ$  latitud,  $6.18^\circ$  longitud coordenadas absolutas,  
 $-4^\circ$  oeste coordenadas relativas

# Distancia

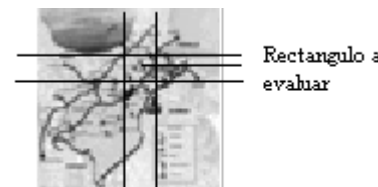


- Sean  $x$ ,  $y$  dos objetos espaciales, la distancia  $D(x,y)$ :
  - ❖  $D(x, y) \geq 0 \quad \forall x,y$
  - ❖  $D(x, x) = 0 \quad \forall x$
  - ❖  $D(x, y) = D(y, x) \quad \forall x,y$
  - ❖  $D(x, y) \leq D(x, z) + D(z, y) \quad \forall x,y,z$
- Información espacial indeterminada (no se exactamente donde) depende de la granularidad del sistema
- Representación física:
  - ❖ 32 bits/dimensión permite una granularidad de 1 decímetro en 2D

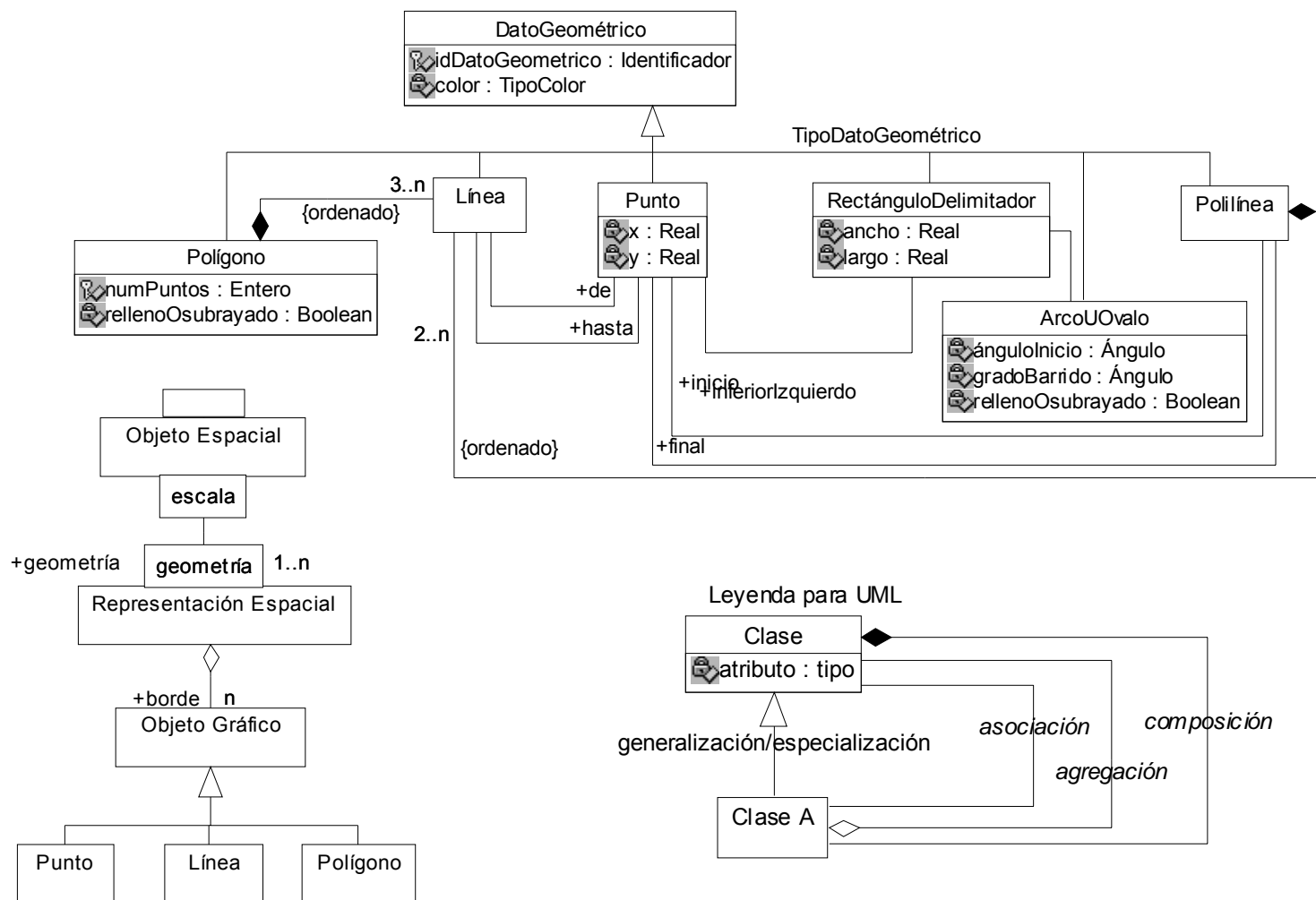
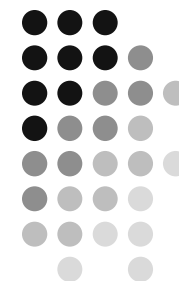
# Objetos espaciales



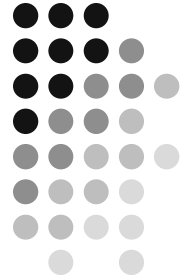
- Contiene además datos no espaciales como nombre de la ciudad, número de habitantes, etc.
- En GIS se almacenan datos espaciales con una representación espacial que pueden ser desplegados gráficamente.
  - ❖ El espacio en GIS se refiere al espacio geográfico donde los objetos geográficos se relacionan
- Tipos de representación:
  - ❖ Matricial y vectorial



# Diagrama de clases para objetos espaciales

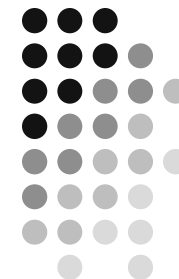


# Relaciones espaciales



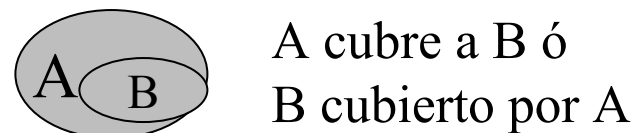
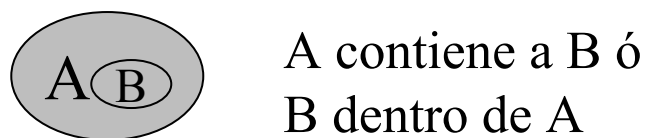
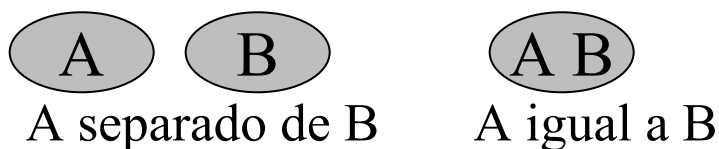
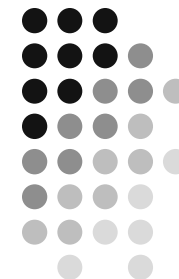
- Las consultas que pueden ser hechas sobre este tipo de datos normalmente involucran relaciones entre los objetos, dadas por su ubicación u ocupación en el espacio
- Entidades espaciales se relacionan de diferentes formas:
  - ❖ Relaciones topológicas (disjuntas, solapamiento, cobertura),
  - ❖ Relaciones direccionales o de posición (norte, arriba, sur, abajo, este, oeste)
  - ❖ Relaciones de proximidad (cerca, lejos, entre).

# Relaciones topológicas



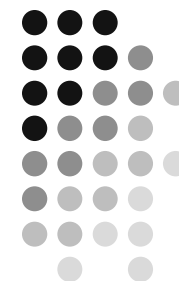
Operador	$b \cap b$	$i \cap i$	$b \cap i$	$i \cap b$
k.separado(m)	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
k.contieneA(m)	$\emptyset$	$\neg \emptyset$	$\emptyset$	$\neg \emptyset$
k.dentro(m)	$\emptyset$	$\neg \emptyset$	$\neg \emptyset$	$\emptyset$
k.toca(m)	$\neg \emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
k.igual(m)	$\neg \emptyset$	$\neg \emptyset$	$\emptyset$	$\emptyset$
k.cubre(m)	$\neg \emptyset$	$\neg \emptyset$	$\emptyset$	$\neg \emptyset$
k.cubiertoPor(m)	$\neg \emptyset$	$\neg \emptyset$	$\neg \emptyset$	$\emptyset$
k.solapa(m)	$\neg \emptyset$	$\neg \emptyset$	$\neg \emptyset$	$\neg \emptyset$

# Relaciones topológicas

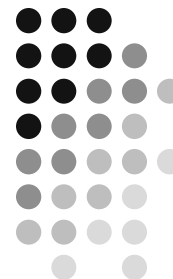


**mbr** es el mínimo  
rectángulo que  
cubre el objeto,  
como **[xi, xs, yi, ys]**

# Relaciones direccionales



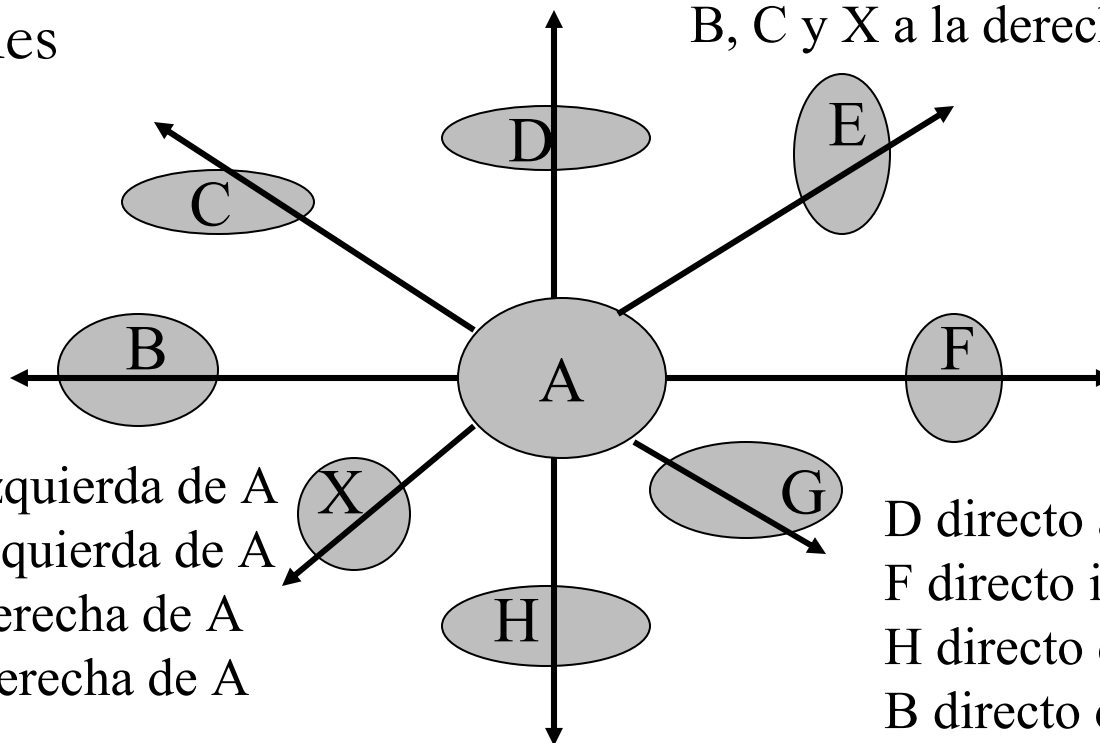
Operador	Relación
k.arriba(m)	$k.y_i \geq m.y_s$
k.abajo(m)	$k.x_i \geq m.x_s$
k.izquierda(m)	$k.y_s \leq m.y_i$
k.derecha(m)	$k.x_s \leq m.x_i$
k.dirArriba(m)	$k.arriba(m) \wedge m.x.cubre(k.x)$
k.dirAbajo(m)	$k.abajo(m) \wedge m.x.cubre(k.x)$
k.dirIzquierda(m)	$k.izquierda(m) \wedge m.y.cubre(k.y)$
k.dirDerecha(m)	$k.derecha(m) \wedge m.y.cubre(k.y)$
k.arribaIzquierda(m)	$k.arriba(m) \wedge k.izquierda(m)$
k.arribaDerecha(m)	$k.arriba(m) \wedge k.derecha(m)$
k.abajoIzquierda(m)	$k.abajo(m) \wedge k.izquierda(m)$
k.abajoDerecha(m)	$k.abajo(m) \wedge k.derecha(m)$



# Relaciones direccionales

## I Relaciones direccionales entre dos objetos espaciales

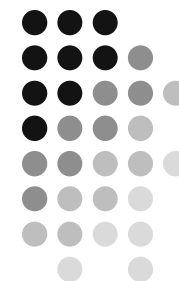
C, D y E arriba de A  
E, F y G a la izquierda de A  
X, H y G debajo de A  
B, C y X a la derecha de A



E arriba y a la izquierda de A  
G abajo y a la izquierda de A  
X abajo y a la derecha de A  
C arriba y a la derecha de A

D directo arriba de A  
F directo izquierda de A  
H directo debajo de A  
B directo derecha de A

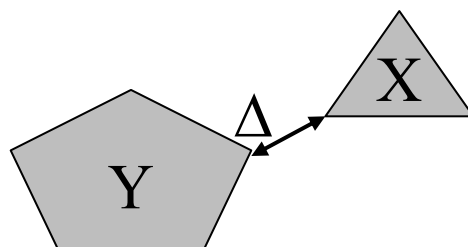
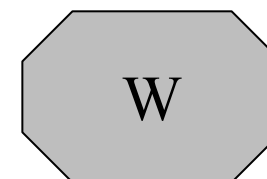
# Relaciones de proximidad



Operador	Relación	P
k.cerca(m)	k.solapa(p)	$(x_{im}-\Delta, x_{sm}+\Delta, y_{im}-\Delta, y_{sm}+\Delta)$
k.lejos(m)	k.separado(p)	
k.entre(m, j)	k.solapa(p)	$(\text{Min}(x_{im}, x_{ij}), \text{Max}(x_{sm}, x_{sj}), \text{Min}(y_{im}, y_{ij}), \text{Max}(y_{sm}, y_{sj}))$

I Relaciones de proximidad  
entre dos objetos  
espaciales

$\Delta$  unidad para  
calcular si un mbr  
solapa objetos cerca  
del objeto buscado



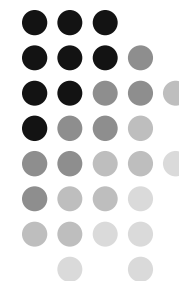
X cerca de Y  
W lejos de Y  
X entre Y y W



```
Class Punto2D
{
    float      x, y;
    char       color[10];
}
```

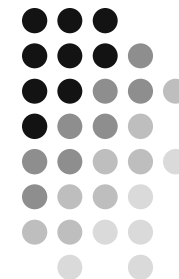
con las operaciones:

# TAD Punto2D



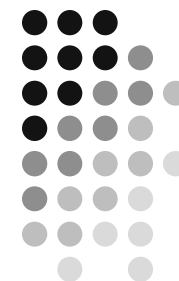
Firma o prototipo de la operación en C++	Descripción
Void leerPunto(Punto2D *);	lee los valores desde el teclado y devuelve las coordenadas del punto
Void cambiaColor(Punto2D, char [])	Cambia el color del punto
Int cuadrante(Punto2D);	Recibe las coordenadas del punto y regresa el valor entero correspondiente al cuadrante donde se encuentra el punto o si está sobre los ejes o en el origen regresa 0.
Void dezplazarX(Punto2D, float , Punto2D *);	Recibe las coordenadas del punto y regresa las coordenadas del punto desplazado en el eje X un valor real dado en el 2do. parámetro
Void dezplazarY(Punto2D, float , Punto2D *);	Recibe las coordenadas del punto y regresa las coordenadas del punto desplazado en el eje Y un valor real dado en el 2do. parámetro
Bool origen(Punto2D);	Recibe las coordenadas del punto y regresa un valor lógico cierto si el punto está en el origen, de lo contrario regresa falso.
float distancia(Punto2D, Punto2D);	Recibe las coordenadas de 2 puntos y regresa la distancia $\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$
Void cartesianasAcilindricas(Punto2D, float &, flota &);	Recibe un punto y regresa sus coordenadas cilíndricas $\rho=\sqrt{x^2+y^2}$ , $\varphi=\text{Arctg}(y/x)$

# TAD Punto2D



Firma o prototipo de la operación en C++	Descripción
Bool ejeX(Punto2D);	Recibe un punto y regresa un valor lógico cierto si el punto está sobre el eje X, sino regresa falso
Bool ejeY(Punto2D);	Recibe un punto y regresa un valor lógico cierto si el punto está sobre el eje Y, sino regresa falso
Bool menorX(Punto2D, Punto2D);	Regresa cierto si el punto del primer parámetro es menor en el eje X al punto del 2do. parámetro
Bool menorY(Punto2D, Punto2D);	Regresa cierto si el punto del primer parámetro es menor en el eje Y al punto del 2do. parámetro
Bool igual(Punto2D, Punto2D);	Regresa cierto si el punto del primer parámetro es igual al punto del 2do. parámetro
Void escribirPunto(Punto2D);	Recibe las coordenadas del punto y lo despliega en pantalla siguiendo el formato (x, y, color)

# TAD Punto3D



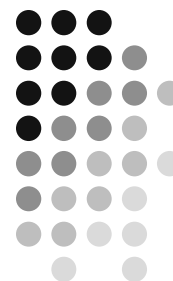
Class Punto3D

```
{
    float    x, y, z;
    char     color[10];
}
```

con las operaciones:

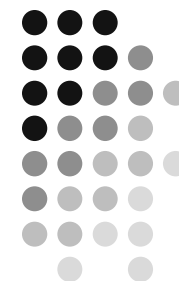
Firma o prototipo de la operación en C++	Descripción
Void leerPunto(Punto3D *);	lee los valores desde el teclado y devuelve las coordenadas del punto
Void cambiaColor(Punto3D, char []);	Cambia el color del punto
Int octante(Punto3D);	Recibe las coordenadas del punto y regresa el valor entero correspondiente al octante donde se encuentra el punto o si está sobre los ejes o en el origen regresa 0.
Void dezplazarX(Punto3D, float , Punto3D *);	Recibe las coordenadas del punto y regresa las coordenadas del punto desplazado en el eje X un valor real dado en el 2do. parámetro
Void dezplazarY(Punto3D, float , Punto3D *);	Recibe las coordenadas del punto y regresa las coordenadas del punto desplazado en el eje Y un valor real dado en el 2do. parámetro
Void dezplazarZ(Punto3D, float , Punto3D *);	Recibe las coordenadas del punto y regresa las coordenadas del punto desplazado en el eje Z un valor real dado en el 2do. parámetro
Bool origen(Punto3D);	Recibe las coordenadas del punto y regresa un valor lógico cierto si el punto está en el origen, de lo contrario regresa falso.

# TAD Punto3D



Firma o prototipo de la operación en C++	Descripción
float distancia(Punto3D, Punto3D);	Recibe las coordenadas de 2 puntos y regresa la distancia $\sqrt{(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2}$
Void cartesianasAcilindricas(Punto3D, float &, float &);	Recibe un punto y regresa sus coordenadas cilíndricas $\rho=\sqrt{x^2+y^2}$ , $\phi=\text{Arctg}(y/x)$
Void cartesianasApolares(Punto3D, float &, float &, float &);	Recibe un punto y regresa sus coordenadas polares $\rho=\sqrt{x^2+y^2+z^2}$ , $\phi=\text{Arctg}(y/x)$ , $\theta=\text{Arctg}(\sqrt{x^2+y^2}/z)$
Bool ejeX(Punto3D);	Recibe un punto y regresa un valor lógico cierto si el punto está sobre el eje X, sino regresa falso
Bool ejeY(Punto3D);	Recibe un punto y regresa un valor lógico cierto si el punto está sobre el eje Y, sino regresa falso
Bool ejeZ(Punto3D);	Recibe un punto y regresa un valor lógico cierto si el punto está sobre el eje Z, sino regresa falso
Bool menorX(Punto3D, Punto3D);	Regresa cierto si el punto del primer parámetro es menor en el eje X al punto del 2do. parámetro
Bool menorY(Punto3D, Punto3D);	Regresa cierto si el punto del primer parámetro es menor en el eje Y al punto del 2do. parámetro
Bool menorZ(Punto3D, Punto3D);	Regresa cierto si el punto del primer parámetro es menor en el eje Z al punto del 2do. parámetro
Bool igual(Punto3D, Punto3D);	Regresa cierto si el punto del primer parámetro es igual al punto del 2do. parámetro
Void escribirPunto(Punto3D);	Recibe las coordenadas del punto y lo despliega en pantalla siguiendo el formato (x, y, z, color)

# TAD Linea2D



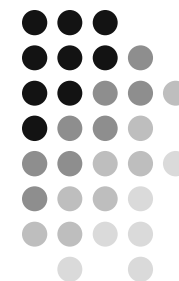
Class Linea2D

```
{
    Punto2D desde, hasta;
    short   grosor;
    char    color[10];
}
```

con las operaciones:

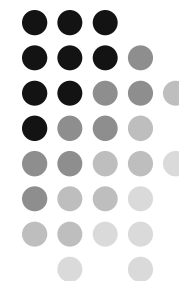
Firma o prototipo de la operación en C++	Descripción
Void leerLinea(Linea2D *);	Lee los valores desde el teclado y devuelve la línea
void cambiaGrosor(Linea2D, short);	Cambia el valor del grosor de la línea
Void cambiaColor(Linea2D, char []);	Cambia el color de la línea
Void dezplazarX(Linea2D, float , Linea2D *);	Recibe la línea y regresa otra línea desplazada en el eje X un valor real dado en el 2do. parámetro
Void dezplazarY(Linea2D, float , Linea2D *);	Recibe la línea y regresa otra línea desplazada en el eje Y un valor real dado en el 2do. parámetro
Bool interseccion(Linea2D, Linea2D);	Regresa cierto si la línea del 1er parámetro cruza la línea del 2do parámetro.
float longitud(Linea2D);	Recibe la línea y regresa su longitud que es la distancia entre sus dos puntos

# TAD Linea2D



Firma o prototipo de la operación en C++	Descripción
Bool arriba(Linea2D, Linea2D);	Regresa cierto si la línea del primer parámetro está arriba de la línea del 2do. parámetro
Bool abajo(Linea2D, Linea2D);	Regresa cierto si la línea del 1er parámetro está debajo de la línea del 2do parámetro
Bool izquierda(Linea2D, Linea2D);	Regresa cierto si la línea del 1er parámetro está a la izquierda de la línea del 2do parámetro
Bool derecha(Linea2D, Linea2D);	Regresa cierto si la línea del 1er parámetro está a la derecha de la línea del 2do. parámetro
Bool perpendicularX(Linea2D);	Regresa cierto si la línea es perpendicular al eje X
Bool perpendicularY(Linea2D);	Regresa cierto si la línea es perpendicular al eje Y
Bool igual(Linea2D, Linea2D);	Regresa cierto si la línea del primer parámetro es igual a la línea del 2do. parámetro
Void escribirLinea(Linea2D);	Recibe la línea y despliega en pantalla siguiendo el formato desde, hacia, grosor y color

# TAD Linea3D



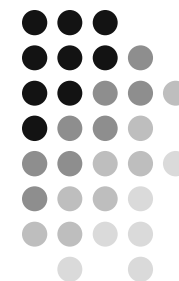
## Class Linea3D

```
{
    Punto3D desde, hasta;
    short   grosor;
    char    color[10];
}
```

con las operaciones:

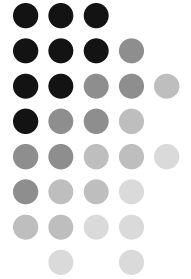
Firma o prototipo de la operación en C++	Descripción
Void leerLinea(Linea3D *);	Lee los valores desde el teclado y devuelve la línea
void cambiaGrosor(Linea3D, short);	Cambia el valor del grosor de la línea
Void cambiaColor(Linea3D, char []);	Cambia el color de la línea
Void dezplazarX(Linea3D, float , Linea3D *);	Recibe la línea y regresa otra línea desplazada en el eje X un valor real dado en el 2do. parámetro
Void dezplazarY(Linea3D, float , Linea3D *);	Recibe la línea y regresa otra línea desplazada en el eje Y un valor real dado en el 2do. parámetro
Void dezplazarZ(Linea3D, float , Linea3D *);	Recibe la línea y regresa otra línea desplazada en el eje Z un valor real dado en el 2do. parámetro

# TAD Linea3D



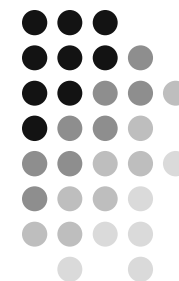
Firma o prototipo de la operación en C++	Descripción
Bool interseccion(Linea3D, Linea3D);	Regresa cierto si la línea del 1er parámetro cruza la línea del 2do parámetro.
float longitud(Linea3D);	Recibe la línea y regresa su longitud que es la distancia entre sus dos puntos
Bool arriba(Linea3D, Linea3D);	Regresa cierto si la línea del primer parámetro está arriba de la línea del 2do. parámetro
Bool abajo(Linea3D, Linea3D);	Regresa cierto si la línea del 1er parámetro está debajo de la línea del 2do parámetro
Bool izquierda(Linea3D, Linea3D);	Regresa cierto si la línea del 1er parámetro está a la izquierda de la línea del 2do parámetro
Bool derecha(Linea3D, Linea3D);	Regresa cierto si la línea del 1er parámetro está a la derecha de la línea del 2do. parámetro
Bool perpendicularX(Linea3D);	Regresa cierto si la línea es perpendicular al eje X
Bool perpendicularY(Linea3D);	Regresa cierto si la línea es perpendicular al eje Y
Bool perpendicularZ(Linea3D);	Regresa cierto si la línea es perpendicular al eje Z

# TAD Linea3D



Firma o prototipo de la operación en C++	Descripción
Bool igual(Linea3D, Linea3D);	Regresa cierto si la línea del primer parámetro es igual a la línea del 2do. parámetro
Void escribirLinea(Linea3D);	Recibe la línea y despliega en pantalla siguiendo el formato desde, hacia, grosor y color

# TAD Rectangulo

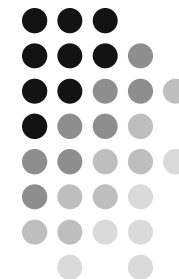


```
Class Rectangulo
{
    Punto2D      iz, de;
    char         color[10];
}
```

con las operaciones:

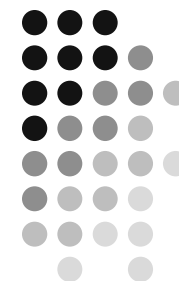
Firma o prototipo de la operación en C++	Descripción
Void leer(Rectangulo *);	Lee los valores desde el teclado y devuelve el rectángulo
Void cambiaColor(Rectangulo, char []);	Cambia el color del rectángulo
float area(Rectangulo);	Regresa el área del rectángulo
Void dezplazarX(Rectangulo, float , Rectangulo *);	Recibe un rectángulo y regresa otro rectángulo desplazado en el eje X un valor real dado por el 2do. parámetro
Void dezplazarY(Rectangulo, float , Rectangulo *);	Recibe un rectángulo y regresa otro rectángulo desplazado en el eje Y un valor real dado por el 2do. parámetro

# TAD Rectangulo



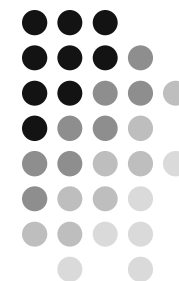
Firma o prototipo de la operación en C++	Descripción
<code>void agranda(Rectangulo, float, Rectangulo *);</code>	Regresa un rectángulo agrandado proporcional-mente, la cantidad indicada en el 2do parámetro
<code>void encoge(Rectangulo, float, Rectangulo *);</code>	Regresa un rectángulo encogido proporcional-mente, la cantidad indicada en el 2do parámetro
<code>Bool dentro(Rectangulo, Punto2D);</code>	Regresa cierto si el punto del 2do parámetro está dentro del rectángulo
<code>Bool igual(Rectangulo, Rectangulo);</code>	Regresa cierto si el rectángulo del 1er parámetro es igual al rectángulo del 2do. parámetro
<code>Bool solapa(Rectangulo, Rectangulo);</code>	Regresa cierto si el rectángulo del 1er parámetro solapa o intersecta al rectángulo del 2do parámetro

# TAD Rectangulo



Firma o prototipo de la operación en C++	Descripción
Bool arriba(Rectangulo, Rectangulo);	Regresa cierto si el rectángulo del 1er parámetro está arriba del rectángulo del 2do parámetro
Bool abajo(Rectangulo, Rectangulo);	Regresa cierto si el rectángulo del 1er parámetro está abajo del rectángulo del 2do parámetro
Bool izquierda(Rectangulo, Rectangulo);	Regresa cierto si el rectángulo del 1er parámetro está a la izquierda del rectángulo del 2do parámetro
Bool derecha(Rectangulo, Rectangulo);	Regresa cierto si el rectángulo del 1er parámetro está a la derecha del rectángulo del 2do parámetro
Void escribir(Rectangulo);	Recibe el rectángulo y lo despliega en pantalla siguiendo el formato (iz, de, color)

# TAD Paralelepipedo



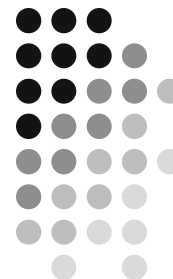
Class Paralelepipedo

```
{
    Punto3D          iz;
    float            ancho, largo, alto;
    char             color[10];
}
```

con las operaciones:

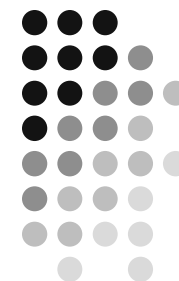
Firma o prototipo de la operación en C++	Descripción
Void leer(Paralelepipedo *);	Lee los valores desde el teclado y devuelve el paralelepípedo
Void cambiaColor(Paralelepipedo, char []);	Cambia el color del paralelepípedo
float volumen(Paralelepipedo);	Regresa el volumen del paralelepípedo
Void dezplazarX(Paralelepipedo, float , Paralelepipedo *);	Recibe un paralelepípedo y regresa otro paralelepípedo desplazado en el eje X un valor real dado por el 2do. parámetro
Void dezplazarY(Paralelepipedo, float , Paralelepipedo *);	Recibe un paralelepípedo y regresa otro paralelepípedo desplazado en el eje Y un valor real dado por el 2do. parámetro
Void dezplazarZ(Paralelepipedo, float , Paralelepipedo *);	Recibe un paralelepípedo y regresa otro paralelepípedo desplazado en el eje Z un valor real dado por el 2do. parámetro

# TAD Paralelepipedo



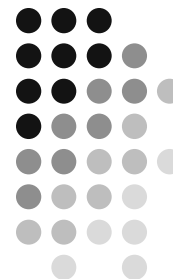
Firma o prototipo de la operación en C++	Descripción
<code>void agranda(Paralelepipedo, float, Paralelepipedo *);</code>	Regresa un paralelepípedo agrandado proporcionalmente, la cantidad indicada en el 2do parámetro
<code>void encoge(Paralelepipedo, float, Paralelepipedo *);</code>	Regresa un paralelepípedo encogido proporcionalmente, la cantidad indicada en el 2do parámetro
<code>Bool dentro(Paralelepipedo, Punto3D);</code>	Regresa cierto si el punto del 2do parámetro está dentro del paralelepípedo
<code>Bool igual(Paralelepipedo, Paralelepipedo);</code>	Regresa cierto si el paralelepípedo del 1er parámetro es igual al paralelepípedo del 2do. parámetro
<code>Bool solapa(Paralelepipedo, Paralelepipedo);</code>	Regresa cierto si el paralelepípedo del 1er parámetro solapa o intersecta al paralelepípedo del 2do parámetro

# TAD Paralelepipedo



Firma o prototipo de la operación en C++	Descripción
Bool arriba(Paralelepipedo, Paralelepipedo);	Regresa cierto si el paralelepípedo del 1er parámetro está arriba del paralelepípedo del 2do parámetro
Bool abajo(Paralelepipedo, Paralelepipedo);	Regresa cierto si el paralelepípedo del 1er parámetro está abajo del paralelepípedo del 2do parámetro
Bool izquierda(Paralelepipedo, Paralelepipedo);	Regresa cierto si el paralelepípedo del 1er parámetro está a la izquierda del paralelepípedo del 2do parámetro
Bool derecha(Paralelepipedo, Paralelepipedo);	Regresa cierto si el paralelepípedo del 1er parámetro está a la derecha del paralelepípedo del 2do parámetro
Bool atras(Paralelepipedo, Paralelepipedo);	Regresa cierto si el paralelepípedo del 1er parámetro está atrás del paralelepípedo del 2do parámetro
Bool adelante(Paralelepipedo, Paralelepipedo);	Regresa cierto si el paralelepípedo del 1er parámetro está adelante del paralelepípedo del 2do parámetro
Void escribir(Paralelepipedo);	Recibe el paralelepípedo y lo despliega en pantalla siguiendo el formato (iz, de, color)

# TAD Triangulo



```

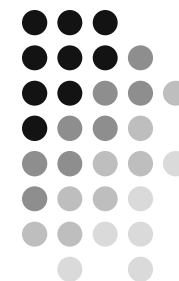
Class Triangulo
{
    Punto2D a, b, c;
    char    color[10];
}

```

con las operaciones:

Firma o prototipo de la operación en C++	Descripción
Void leerTriangulo(Triangulo *);	Lee los valores desde el teclado y devuelve el triángulo
Void cambiaColor(Triangulo, char []);	Cambia el color del triángulo
Bool valido(Triangulo);	Valida si se puede formar un triángulo con los tres puntos
Void dezplazarX(Triangulo, float , Triangulo *);	Recibe un triángulo y regresa otro triángulo desplazado en el eje X un valor real dado en el 2do. parámetro
Void dezplazarY(Triangulo, float , Triangulo *);	Recibe un triángulo y regresa otro triángulo desplazado en el eje Y un valor real dado en el 2do. parámetro

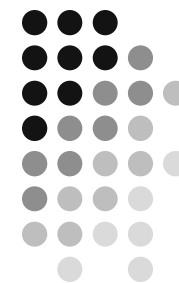
# TAD Triangulo



Firma o prototipo de la operación en C++	Descripción
void tipo(Triangulo, char []);	Regresa el tipo de triángulo: isósceles, rectángulo, equilátero, etc.
float area(Triangulo);	Regresa el área del triángulo
float altura(Triangulo);	Regresa la altura del triángulo
Bool igual(Triangulo, Triangulo);	Regresa cierto si el triángulo del 1er parámetro es igual al triángulo del 2do. parámetro
Bool arriba(Triangulo, Triangulo);	Regresa cierto si el triángulo del 1er parámetro está arriba del triángulo del 2do parámetro
Bool abajo(Triangulo, Triangulo);	Regresa cierto si el triángulo del 1er parámetro está abajo del triángulo del 2do parámetro
Bool izquierda(Triangulo, Triangulo);	Regresa cierto si el triángulo del 1er parámetro está a la izquierda del triángulo del 2do parámetro
Bool derecha(Triangulo, Triangulo);	Regresa cierto si el triángulo del 1er parámetro está a la derecha del triángulo del 2do parámetro
Void dibujar(Triangulo);	Dibuja el triángulo sin color
Void escribirTriangulo(Triangulo);	Recibe un triángulo y lo despliega en pantalla siguiendo el formato a, b, c y color

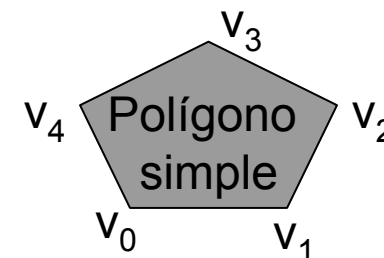


# Triangulación de polígonos



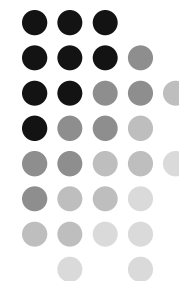
- Polígono simple: Sea  $v_0, v_1, \dots, v_{n-1}$   $n$  puntos en el plano  $XY$ , denominados vértices. Sea  $e_0 = v_0v_1$ ,  $e_1 = v_1v_2, \dots, e_i = v_iv_{i+1}, \dots, e_{n-1} = v_{n-1}v_0$ ,  $n$  segmentos conectando los  $n$  puntos. Dichos segmentos bordean un polígono si y solo si

1.  $e_i \cap e_{i+1} = v_{i+1} \quad \forall i=0, \dots, n-1$
2.  $e_i \cap e_j = \emptyset \quad \forall j \neq i+1$



- Los vértices se listan en el sentido contra reloj definiendo una travesía del perímetro (boundary traversal)

# Teorema de Jordan

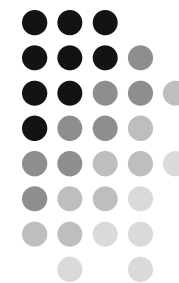


- Cada curva simple cerrada en el plano divide éste en dos componentes
  - ❖ Interior: está limitado, justifica la definición de que un polígono es una región acotada por una colección de segmentos
  - ❖ Exterior: no está limitado
- El perímetro  $\partial$  de un polígono  $P$ , donde  $\partial P \subseteq P$



Si  $P$  no es simple, se cumple la condición 1 pero no la 2,  $e_i \cap e_j = \emptyset \forall j \neq i+1$

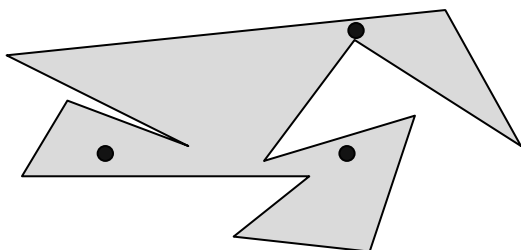
# Teorema de la galería de arte



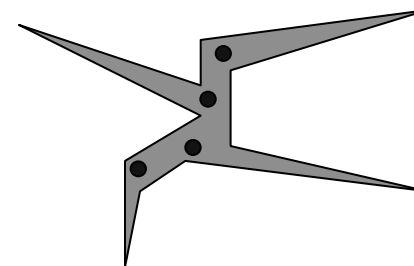
- Definición del problema (Klee, 1973): Considere una galería de arte cuyo plano se modela con un polígono de  $n$  vértices ¿Cuántos guardias se necesitan para vigilar un cuarto?

Misma  
formulación  
para bombillos  
de alumbrado  
del cuarto

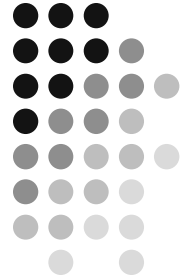
- Cada guardia se considera un punto fijo que puede mirar en cada dirección con  $2\pi$  de rango de visibilidad y no puede mirar a través de las paredes



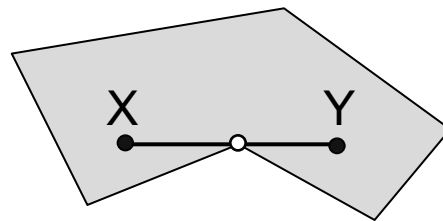
Dos polígonos de  
 $n=12$  vértices



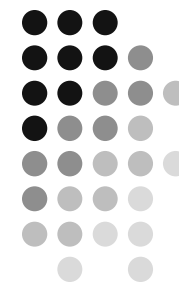
# Visibilidad



- $X$  puede ver  $Y$  ( $Y$  es visible para  $X$ ) si y solo si el segmento  $XY$  no está en el exterior del polígono:  
 $XY \subseteq P$
- $X$  tiene una visión clara de  $Y$  si  $XY \subseteq P$  y  
 $XY \cap \partial P \subseteq \{X, Y\}$
- Un conjunto de guardias cubre a  $P$  si cada punto  $X$  en  $P$  es visible para algún guardia
- Los guardias no se bloquean su visibilidad



# Máximo del mínimo

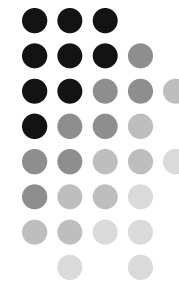


- Encontrar el máximo en todos los polígonos de  $n$  vértices, del mínimo número de guardias necesarios para cubrir el polígono
- Sea  $g(P)$  el mínimo número de guardias necesarios para cubrir  $P$

$g(P) = \min_S | \{S: S \text{ cubre } P\} |$ , donde  $S$  es un conjunto de puntos y  $|S|$  es la cardinalidad de  $S$

- Sea  $P_n$  un polígono de  $n$  vértices, el problema de Klee es determinar  $G(n) = \max_{P_n} g(P_n) \forall P_n$

# Exploración empírica



- Suficiencia de  $n$ : al menos 1 guardia siempre es necesario, a lo sumo 1 guardia por cada vértice

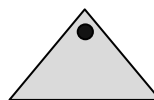
- ❖ Límite inferior:  $1 \leq G(n)$

- ❖ Límite superior:  $G(n) \leq n$

Pero en 3D este límite superior no cubre un poliedro

- Necesidad de un  $n$  pequeño:

- ❖ Triángulos:  $G(3) = 1$

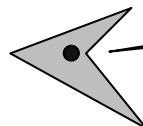


- ❖ Cuadriláteros:  $G(4) = 1$

- Convexos

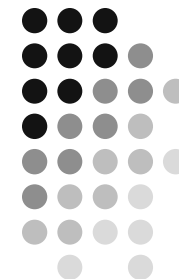


- Con 1 vértice reflejo




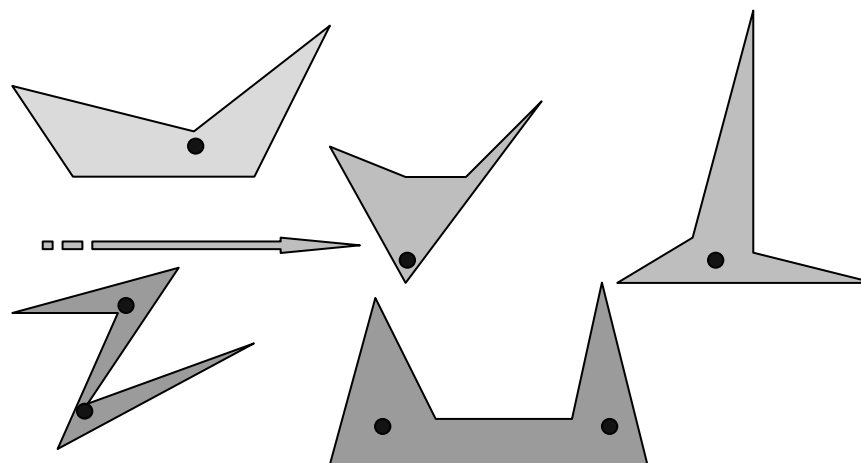
Vértice reflejo o concavo si su ángulo interno es estrictamente mayor que  $\pi$

# Exploración empírica



❖ Pentágonos:  $G(5) = 1$

- Convexo 
- Con 1 vértice cóncavo
- Con 2 vértices cóncavos



❖ Hexágonos:  $G(6) = 2$

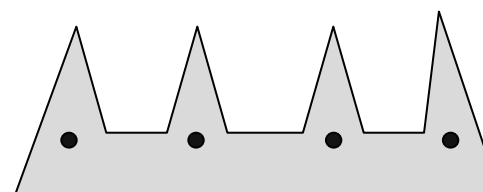
➤ Generalización:

Un *prong* está compuesto de 2 aristas y *prongs* adyacentes están separados con 1 arista

Una figura con  $k$  *prongs* tiene

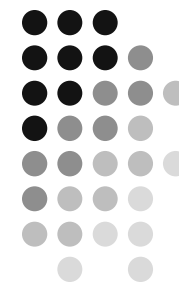
$n=3k$  aristas o vértices

$\lfloor n/3 \rfloor \leq G(n)$  para  $n=3k$



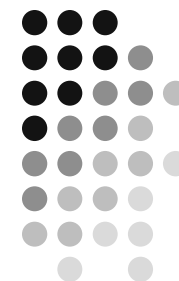


# Prueba de suficiencia de Fisk

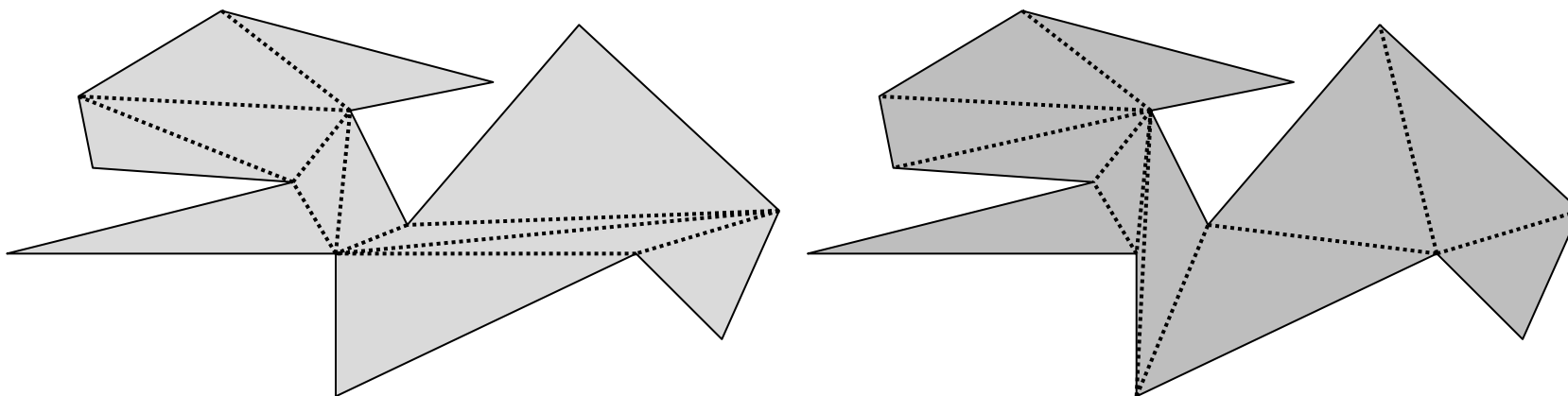


- $G(n) = \lfloor n/3 \rfloor$  por inducción
  - ❖ Chvátal, 1975 y luego Fisk, 1978, depende de la división de  $P$  en triángulos
- Diagonal de  $P$ : es el segmento entre  $a$  y  $b$ , 2 de sus vértices que son claramente visibles uno de otro, esto es: segmento  $ab$  con  $\partial P$  es exactamente  $\{a, b\}$
- Dos diagonales no se cruzan si su intersección es un subconjunto de sus vértices finales, no comparten puntos interiores

# Prueba de suficiencia de Fisk

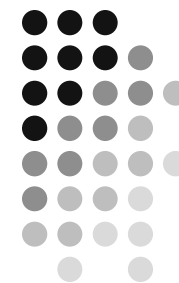


- Triangulación de  $P$ : división de  $P$  con todas las diagonales posibles que no se crucen, tal que el interior de  $P$  se divida en triángulos

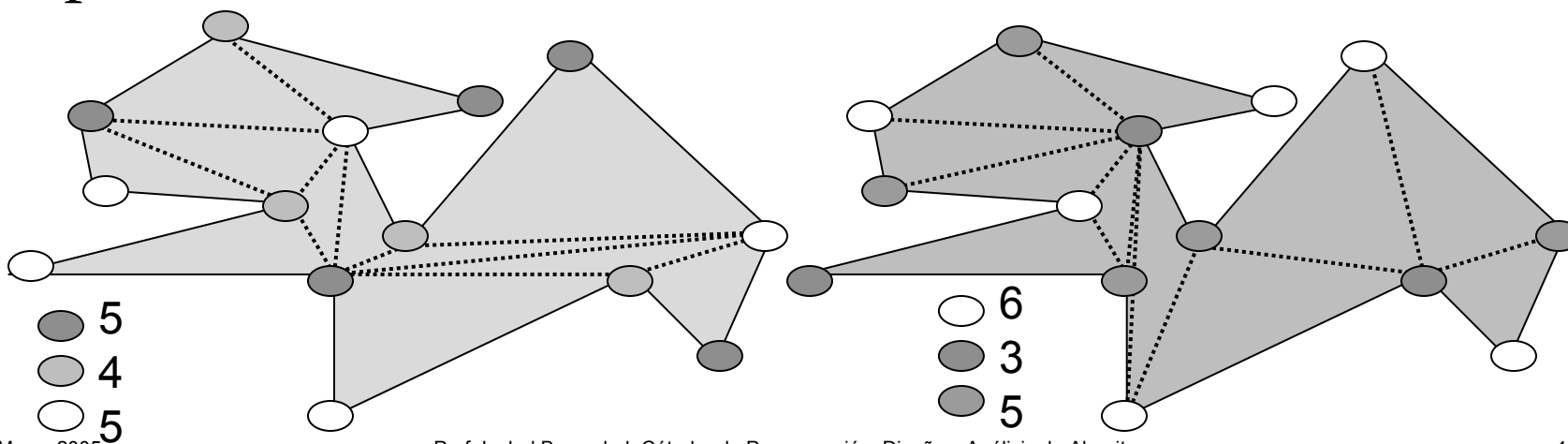


Dos triangulaciones de un polígono de  $n=14$  vértices

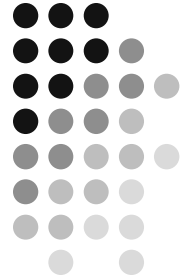
# Prueba de suficiencia de Fisk



- Prueba de suficiencia de  $\lfloor n/3 \rfloor$  guardias para cualquier polígono
- Primer paso: Triangular P
- Segundo paso: mostrar que el grafo de sus nodos puede ser 3-coloreado

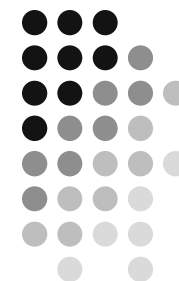


# Grafo 3-coloreado

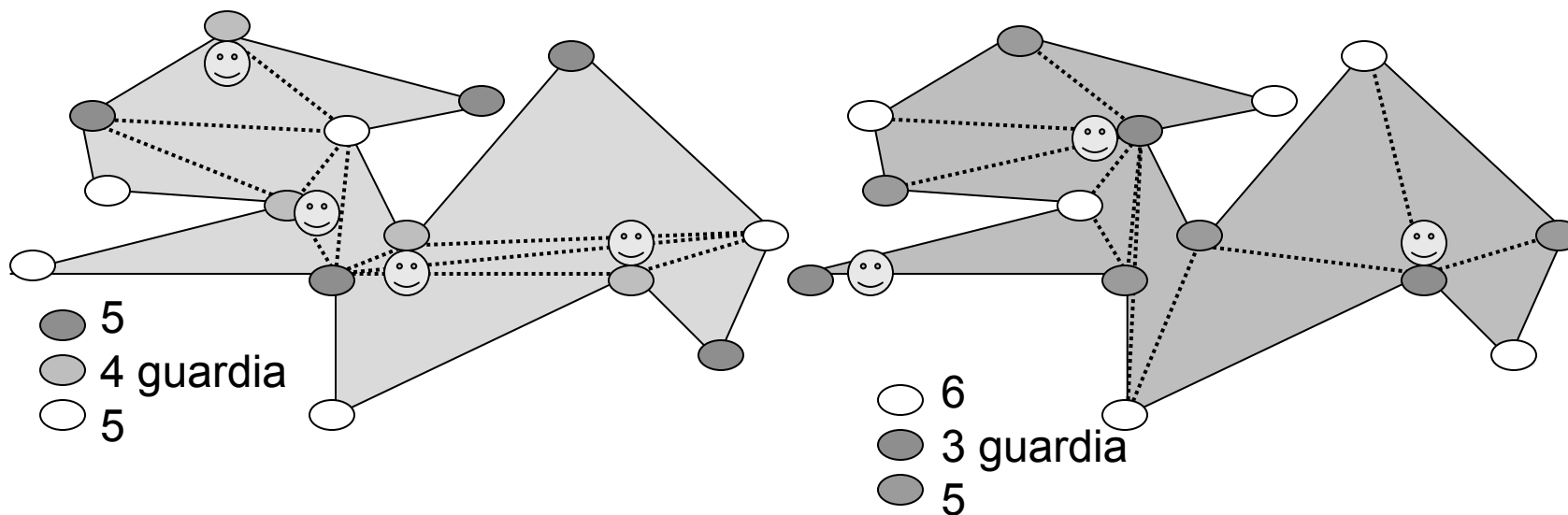


- Sea un grafo  $G$  asociado con la triangulación de un polígono
- Un  $k$ -coloreado de  $G$  es una asignación de  $k$  colores a los nodos de  $G$  tal que no hay dos nodos conectados por una arista que tengan el mismo color
- Fisk asegura que cada  $G$  producto de la triangulación puede ser 3-coloreado
- Se comienza con cualquier triángulo, se colorean sus nodos arbitrariamente y en el resto de los triángulos sus nodos quedan completamente definidos, no hay escogencia libre

# Prueba de suficiencia de Fisk

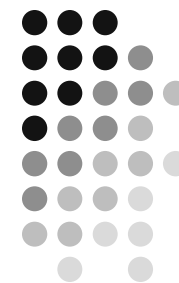


- Tercer paso: colocación de los guardias en todos los vértices que tengan el mismo color, garantizando así la visibilidad en P





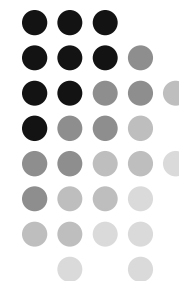
# Prueba de suficiencia de Fisk



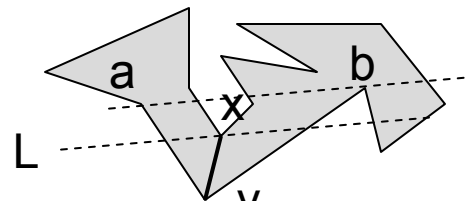
- Cuarto paso:  $n$  nodos y  $k$  colores
- Principio del casillero: si  $n$  objetos se colocan en  $k$  casillas, entonces al menos una casilla debe contener no más de  $n/k$  objetos
  - ❖ si cada una de las  $k$  casillas contiene más de  $n/k$  objetos, el número total de objetos excede a  $n$ .
  - ❖ Un color no puede ser usado más de  $n/3$  veces
- Coloque los guardias en los nodos coloreados con el mismo color menos frecuentemente usado
- Se garantiza que se cubre  $P$  con no más de  $G(n) = \lfloor n/3 \rfloor$  colores



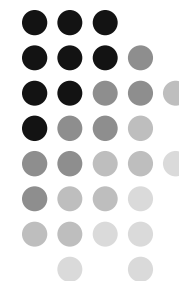
# Triangulación



- Lema Meisters: Cada polígono de  $n \geq 4$  vértices tiene una diagonal
  - ❖ Prueba: Sea  $V$  un vértice convexo,  $a$  y  $b$  sus vértices adyacentes. Si  $ab$  es una diagonal, entonces se demostró. Si  $ab$  no es una diagonal, entonces  $ab$  está en el exterior de  $P$  o ella intersecta  $\partial P$ . Como  $n > 3$ , el triángulo  $avb$  contiene al menos otro vértice  $x$  diferente de  $a$ ,  $b$  o  $v$ . Sea  $x$  el más cercano a  $v$ , cuya distancia se mide perpendicular a la línea  $ab$ , así  $x$  es el primer vértice en  $avb$  que toca la línea  $L$  paralela a  $ab$ , moviéndose desde  $v$  hacia  $ab$ . Entonces  $vx$  es una diagonal de  $P$ , que no puede intersectar  $\partial P$  excepto en  $x$  y en  $v$ .

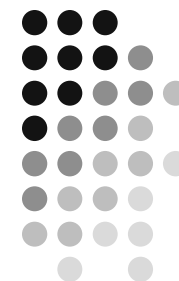


# Triangulación



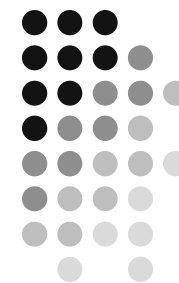
- Teorema triangulación: Cada polígono  $P$  de  $n$  vértices puede dividirse en triángulos mediante la inclusión de 0 o más diagonales
  - ❖ Prueba: por inducción sobre  $n$
  - ❖ Si  $n=3$ ,  $P$  es un triángulo, prueba trivial
  - ❖ Si  $n \geq 4$ , sea  $d=ab$  una diagonal de  $P$ ,  $d$  divide  $P$  en dos polígonos, cada uno con menos de  $n$  vértices, puesto que no se anexan vértices, se aplica la hipótesis nuevamente a los dos subpolígonos.

# Propiedades de las triangulaciones

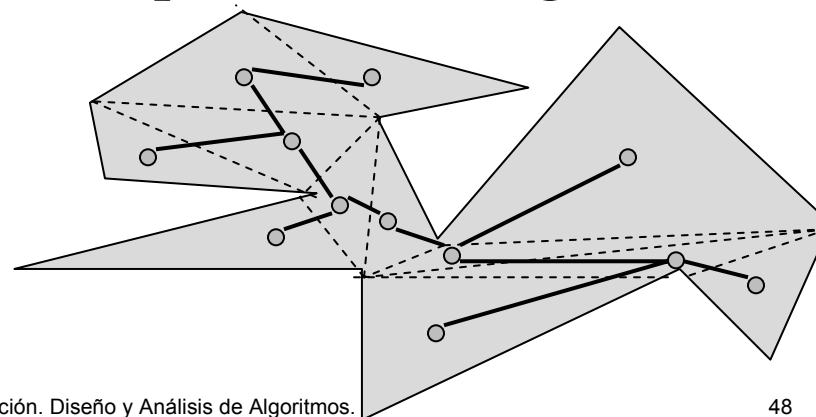


- Lema número de diagonales: Cada triangulación de un polígono  $P$  de  $n$  vértices usa  $n-3$  diagonales y consiste de  $n-2$  triángulos.
  - ❖ Prueba: por inducción sobre  $n$
  - ❖ Si  $n=3$ , caso trivial
  - ❖ Si  $n \geq 4$ ,  $P$  se divide en  $P_1$  y  $P_2$  con la diagonal  $d=ab$  y con  $n_1$  y  $n_2$  vértices cada uno. Se tiene que  $n_1+n_2=n+2$ . Aplicando la hipótesis de inducción, los subpolígonos tendrán  $(n_1-3)+(n_2-3)+1=n-3$  diagonales, y  $(n_1-2)+(n_2-2)=n-2$  triángulos.

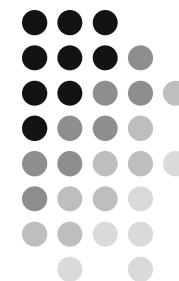
# Triangulación dual



- Corolario suma de ángulos: La suma de los ángulos internos de un polígono de  $n$  vértices es  $(n-2)\pi$ .
  - ❖ Prueba: Hay  $n-2$  triángulos y cada uno contribuye con  $\pi$  para los ángulos internos.
- La triangulación dual  $T$  de un polígono  $P$  es un grafo con un nodo asociado a cada triángulo y una arista entre dos nodos si y sólo si sus triángulos comparten una diagonal

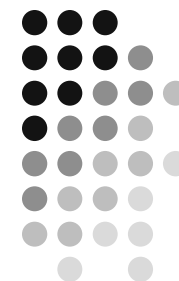


# Triangulación dual



- Lema triangulación dual: la triangulación dual  $T$  es un árbol donde el grado de incidencia de cada nodo es a lo sumo 3.
  - ❖ Prueba: inmediata, dado que cada triángulo tiene a lo sumo tres lados compartidos
- Los nodos de grado de incidencia 1 son las hojas
- Los nodos de grado de incidencia 2 son nodos ramas
- $T$  es un árbol binario cuya raíz es cualquier nodo
- Tres vértices consecutivos  $a$ ,  $b$  y  $c$  de un polígono  $P$  forman una oreja de  $P$  si  $ac$  es una diagonal y  $b$  es la punta de la oreja.
- Dos orejas no se solapan si el interior de sus triángulos son disjuntos

# Prueba de 3-colores



- Teorema dos orejas de Meisters: Cada polígono de  $n \geq 4$  vértices tiene al menos 2 orejas que no se solapan
  - ❖ Prueba: un nodo hoja corresponde a una oreja. Un árbol de 2 o más nodos debe tener al menos 2 hojas
- Teorema 3-colores: el grafo de la triangulación de un polígono  $P$  puede ser 3-coloreado
  - ❖ Prueba: por inducción sobre  $n$ . Un triángulo puede colorearse con 3 colores
  - ❖ Si  $n \geq 4$ ,  $P$  tiene una oreja  $abc$ , si se elimina la oreja de  $P$  se tiene  $P'$ , se reemplaza la secuencia  $abc$  en  $\partial P$  con  $ac$  en  $\partial P'$ .  $P'$  tiene ahora  $n-1$  vértices. Se aplica la hipótesis de 3-colores a  $P'$ , coloree  $b$  con un color diferente a los usados en  $a$  y  $c$ .