Problema de la Galería de Arte

Báez Brenda, Bertino Ariel

Facultad de Ciencias Exactas

Trabajo Final Algoritmos II

Figure: Museo de Arte de Denver, de Daniel Libeskind

Enunciado del problema

En 1973 el matemático norteamericano Victor Klee le propuso a su colega Václav Chvátal el conocido como problema de la galería de arte, que pertenece al área de las matemáticas que recibe el nombre de "geometría combinatoria":

¿Cuál es el mínimo número de guardas, o cámaras de vigilancia, que se necesitan para vigilar una galería de arte?

¿Cómo modelamos matemáticamente una galería de arte?

- Es un polígono simple del plano (el problema es bidimensional) formado por n lados, es decir, la galería de arte está rodeada de una cadena cerrada de paredes (segmentos en el plano) que no se intersecan entre sí. Un cuadrado, un octógono o una estrella de seis puntas son ejemplos de polígonos simples del plano.
- La vigilancia de la galería la llevan a cabo guardas o cámaras de vigilancia que cubren cualquier dirección (es decir, 360°). Su visión está limitada únicamente por paredes.

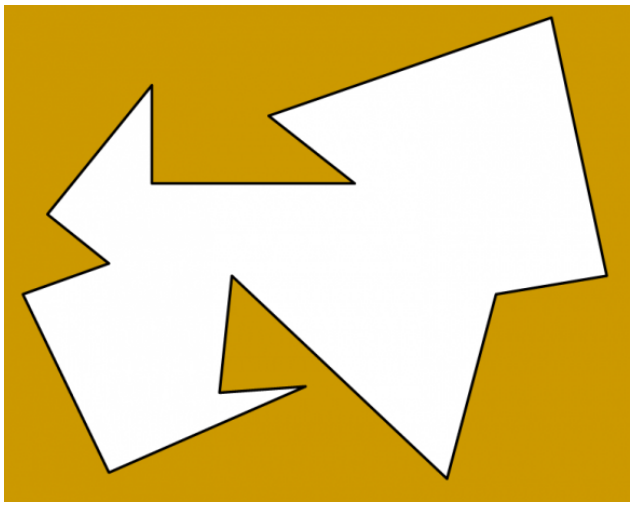


Figure: Polígono simple

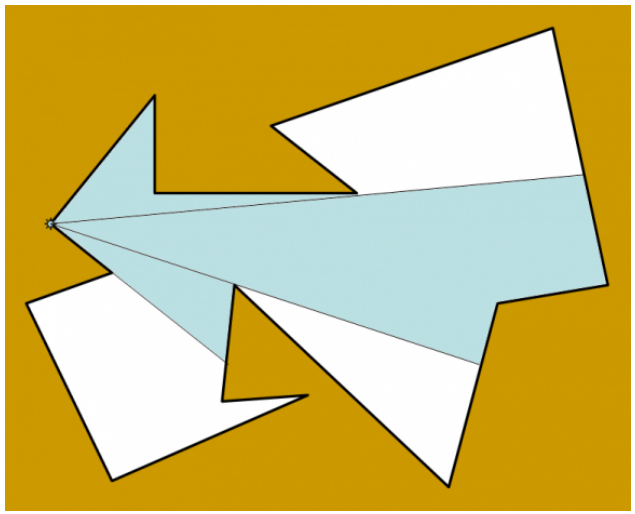


Figure: Visión de una cámara

Solución

En 1978, el matemático Steve Fisk, formuló una demostración que consta de tres pasos

- Triangulación
- Coloreo
- Colocación de las guardas o cámaras

Triangulación. La galería poligonal se puede triangular (es decir, dividir el polígono en triángulos), mediante n-3 diagonales. Por lo tanto, los vértices de la triangulación son los mismos que los del polígono.

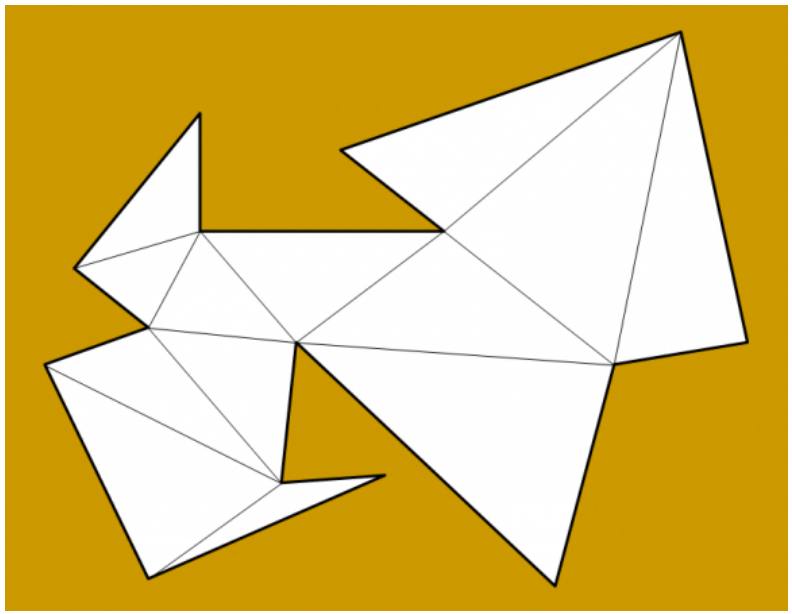


Figure: Triangulación

Coloreo. Se pueden colorear los vértices de la triangulación con tres colores de forma que no haya dos vértices adyacentes con el mismo color (cada vértice de los triángulos que forman la triangulación tendrá un color distinto).

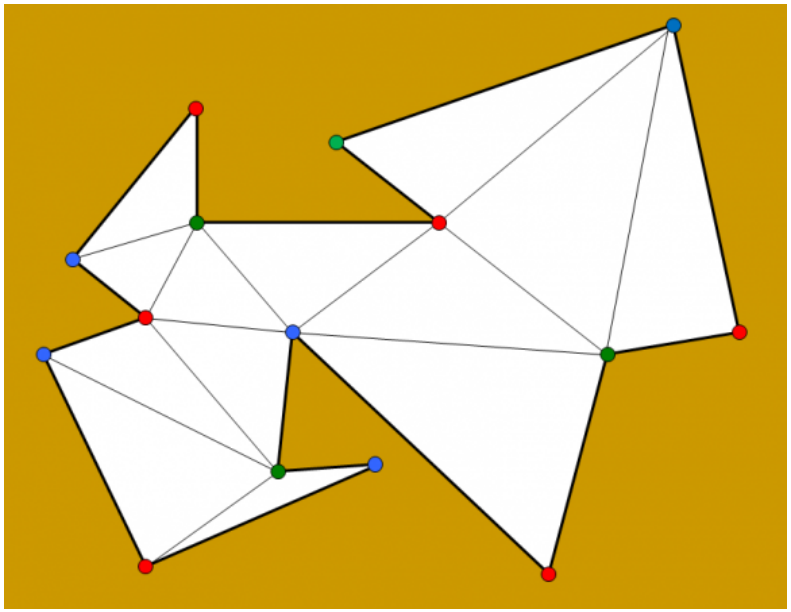


Figure: Coloreo

Colocación de las cámaras. Teniendo en cuenta que hay n vértices, existe por lo menos un color que colorea como mucho n/3 vértices (puede ser exactamente n/3 (función piso) vértices o incluso menos, como se muestra en los siguientes ejemplos). Esos vértices son los elegidos para colocar las cámaras. Además, como cada triángulo tiene un vértice de cada color, todos los triángulos son vigilados por cualquiera de los colores, en particular el elegido.

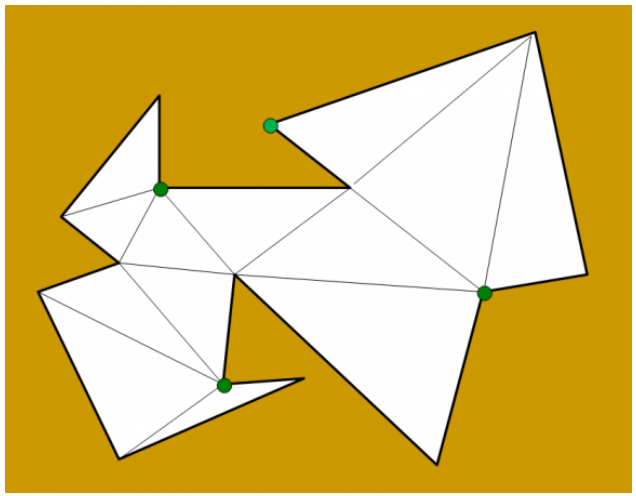


Figure: Colocación de las guardas o cámaras

Implementación

- Explicaremos los algoritmos elegidos.
- Detallaremos las estructuras elegidas.
- Recorreremos el camino de la implementación de cada uno de los tres pasos para resolver este problema.

Algoritmos elegidos

- Para el caso de la triangulación la técnica de resolución elegida fue la triangulación por medio de polígonos monótonos.
- Resolvimos el coloreo, con un algoritmo de coloreo heurístico.
- La colocación de las guardas fue sencilla al ya tener el coloreo implementado.

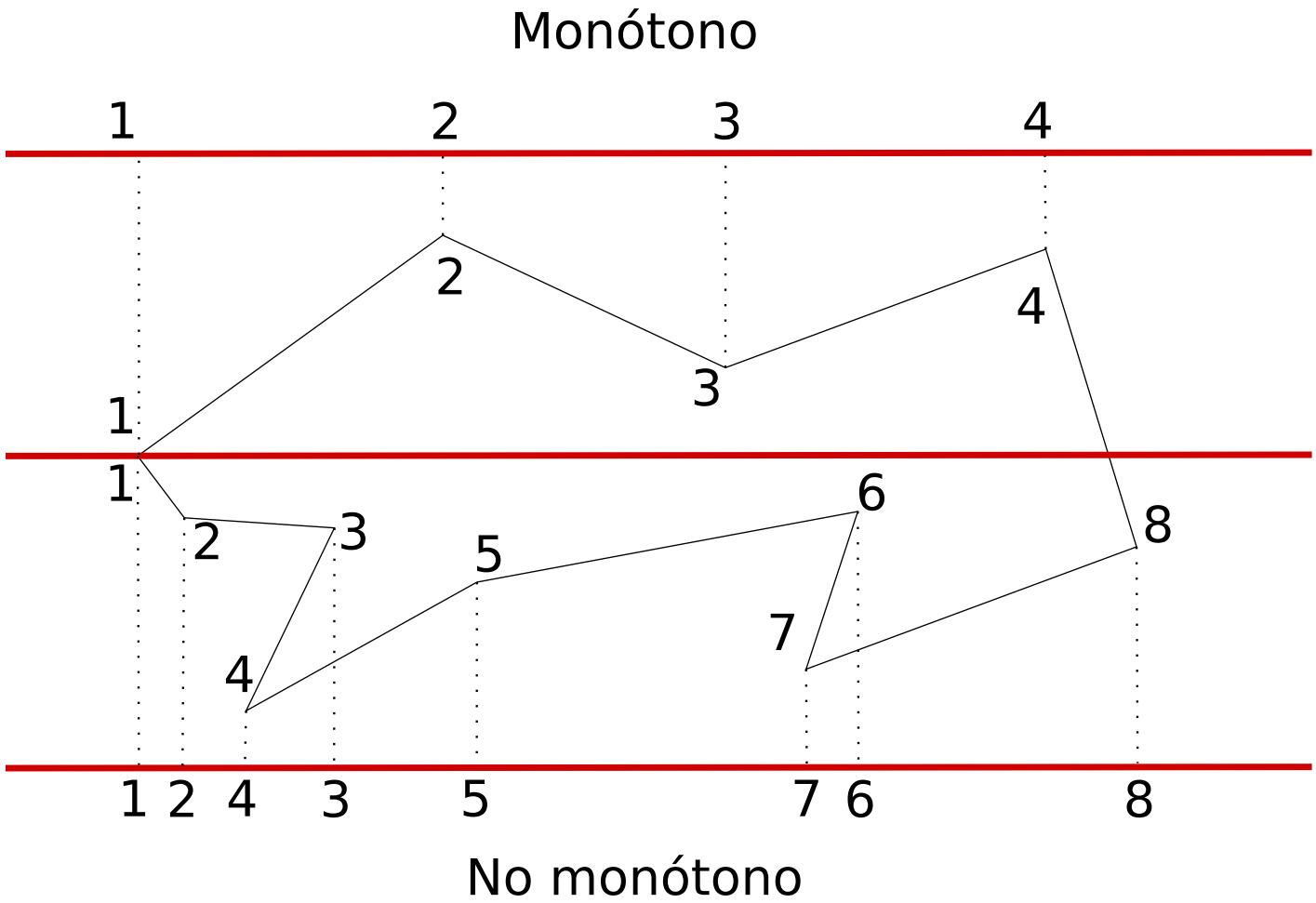
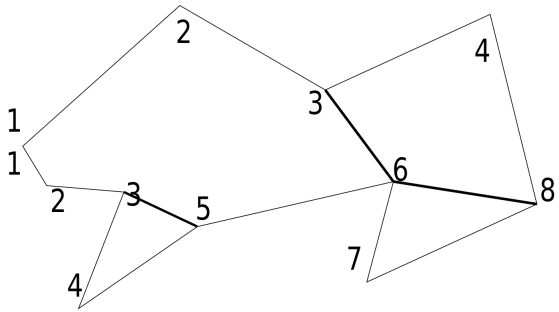
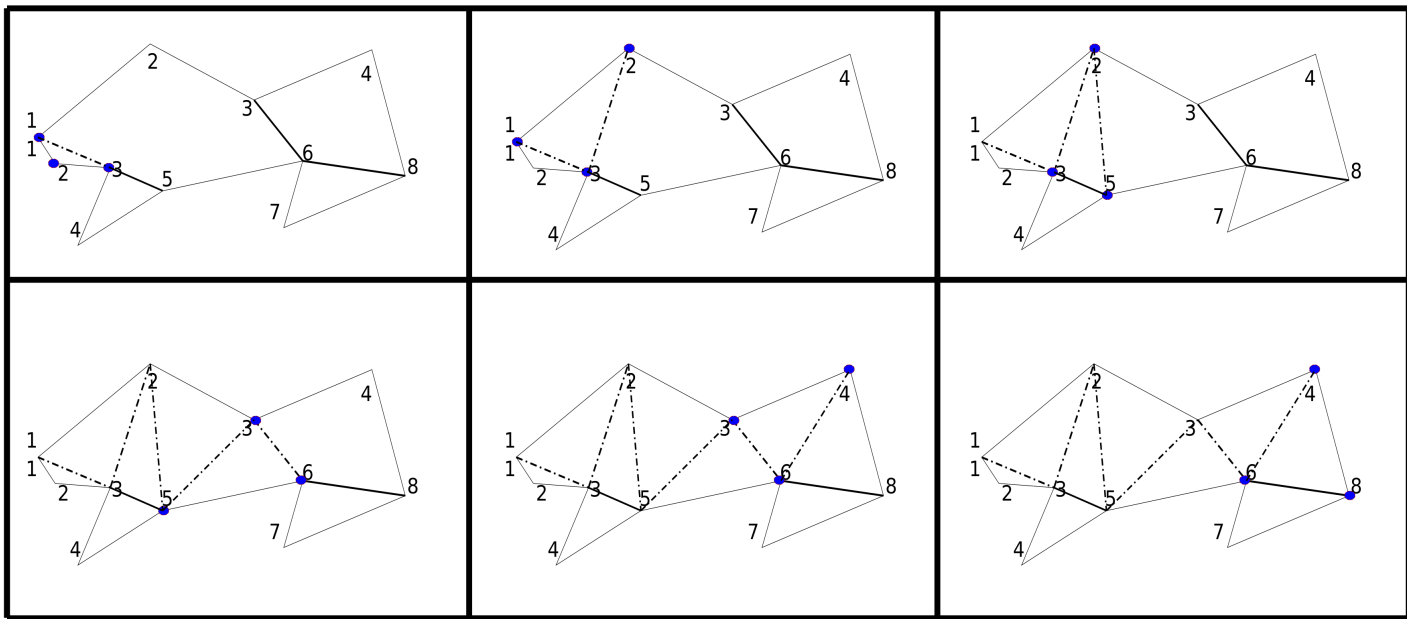


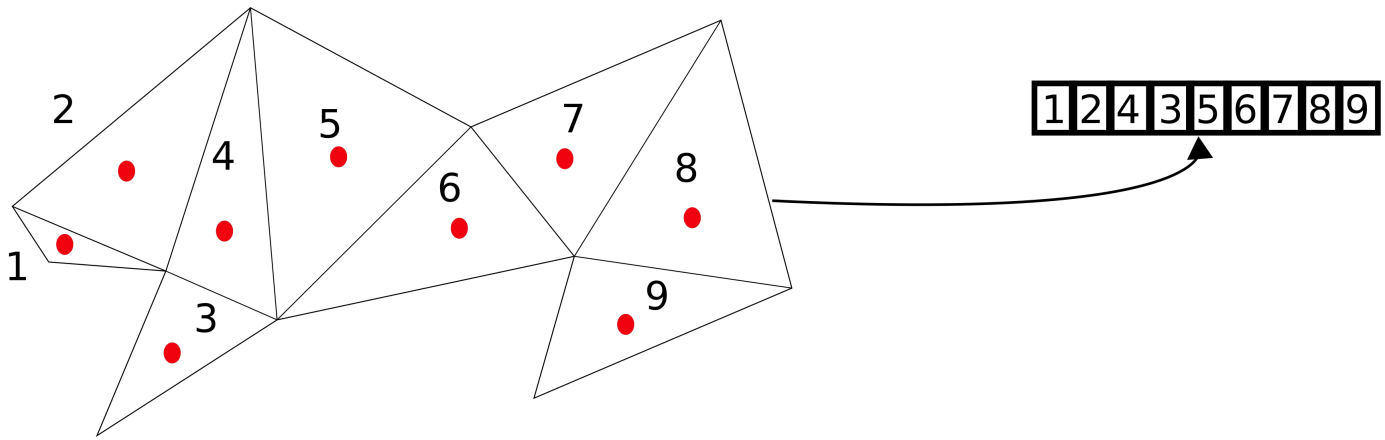
Figure: Ejemplo

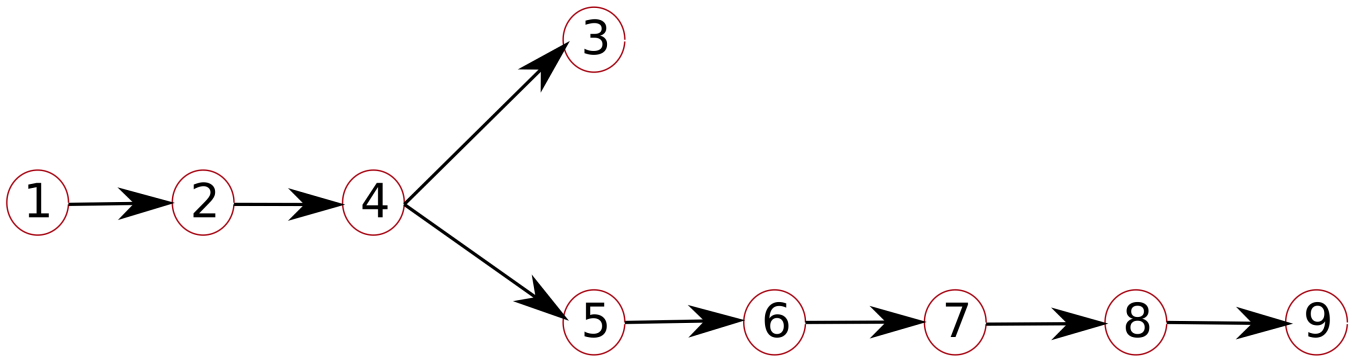
Triangulación





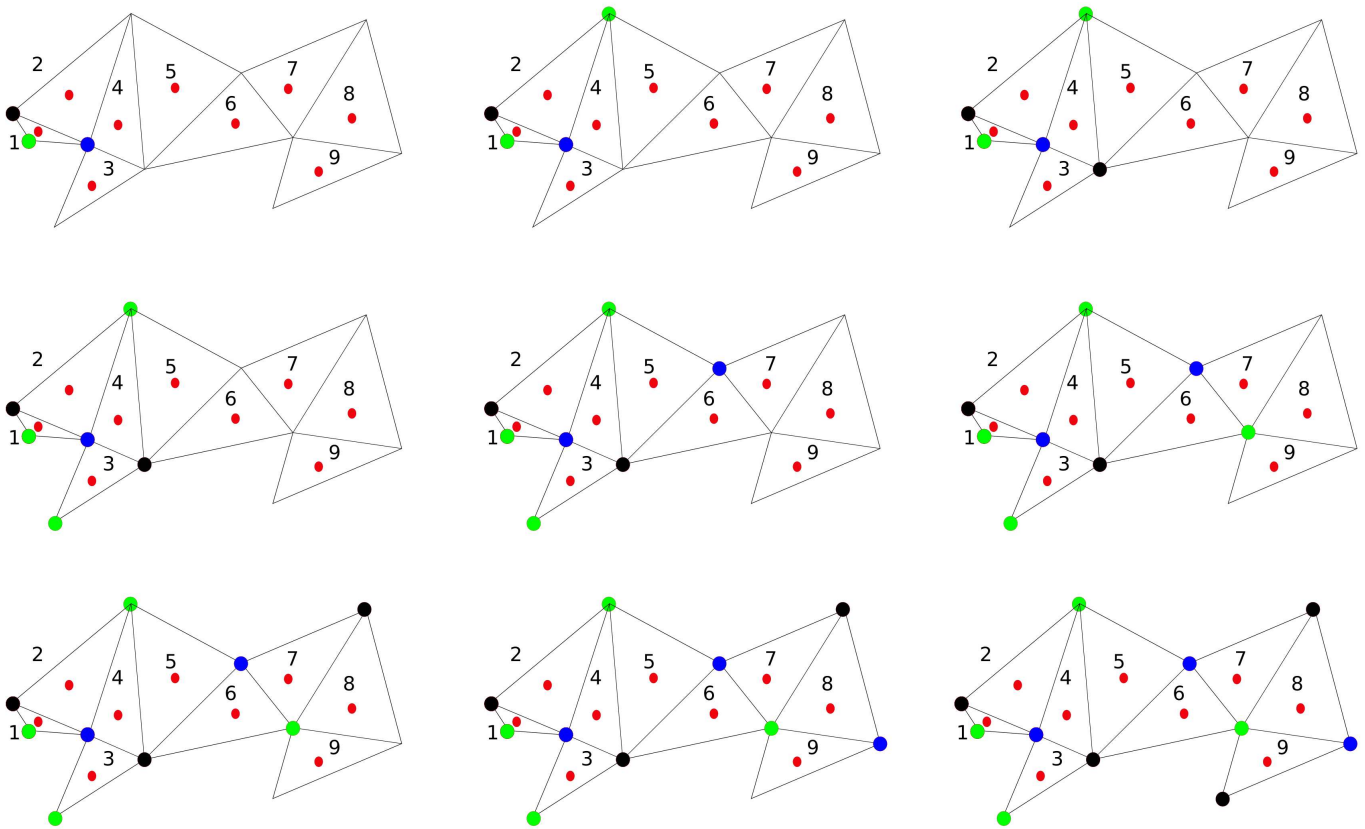
Grafo dual y DFS (pre-coloreo)



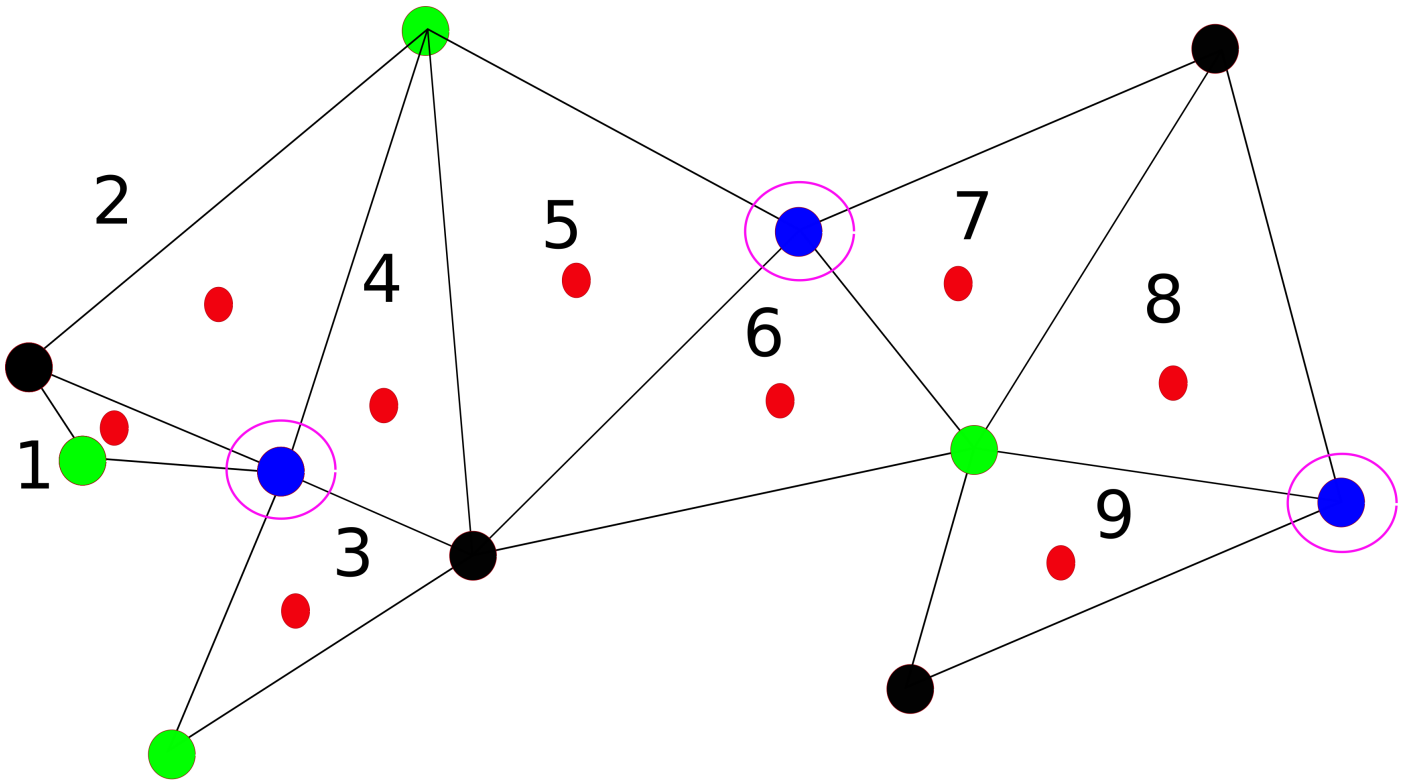


Coloreo

1 2 4 3 5 6 7 8 9



Colocación de guardias



Estructuras elegidas

Creamos una clase Punto

- Almacena las coordenadas x e y (double)
- Tiene un identificador (unsigned int)
- El tipo de punto que es (se detallará en la triangulación) (enum)
- Un campo color (unsigned int)
- Un campo guardia (unsigned int)

Estructuras elegidas

Creamos una clase Lado

- Almacena dos puntos y su tipo (INSERT O INPUT)
- Tiene un identificador (unsigned int)
- Un campo helper (unsigned int)

Estructuras elegidas

Creamos una clase BDMFile

- Se encarga de leer el archivo de entrada y luego devolver las soluciones en nuevos archivos y crear estructuras que luego necesitará el polígono
- Almacena un mapa de puntos (es decir el id del punto y el punto en si mismo)
- Almacena un mapa de lados

Estructuras elegidas

Creamos una clase Polygon

- Hereda de BDMFile sus atributos
- Es el encargado de realizar la triangulación, el coloreo y la colocación de guardias
- Un árbol binario de lados y un campo de orden (se explicará en la triangulación)
- Una lista de los polígonos monótonos obtenidos
- Una lista con los triangulos obtenidos

Estructuras elegidas

Algunas aclaraciones más

- Tanto lados como puntos están almacenados en mapas con ids, entonces el resto de las estructuras no guardan estas clases si no sus identificadores para simplificar los algoritmos
- Ej: un triangulo es una lista con 3 ids de puntos
- Tenemos también un struct llamado NodoGrafo que guarda un triangulo, su centro de masa y tres ids de sus vecinos

Solución

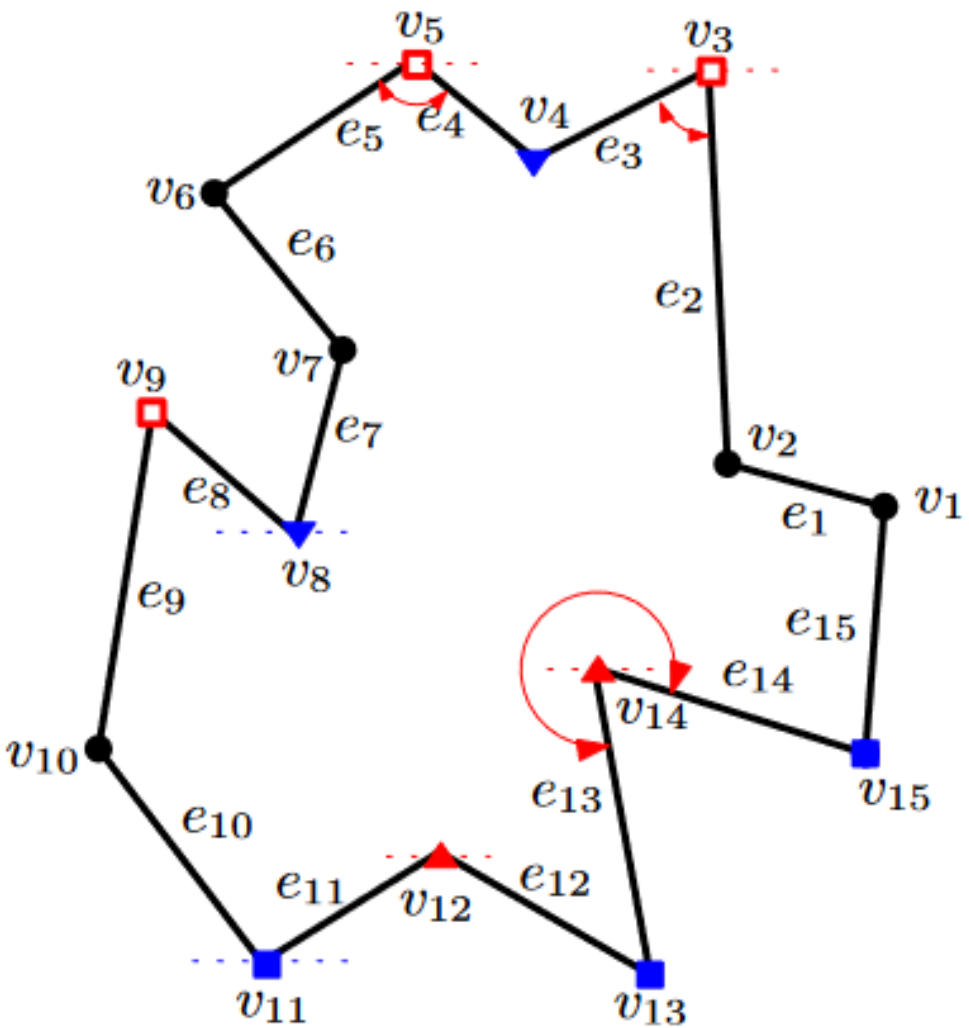
El primer paso es construir nuestro polígono

- Dado el archivo se llama desde el main al constructor de polígono
- El mismo por un lado llama a su constructor padre (BDMFile), el cual lee el archivo y guarda todos los puntos y lados en un mapa con identificador. El tipo de los puntos es DESCONOCIDO y de los lados es tipo INPUT
- Luego el polígono con el mapa de puntos ya creado indica qué tipo de vértice es

Solución

Al recorrer el mapa de puntos, se guarda el valor anterior y siguiente

- Si el punto anterior es mayor y el siguiente es menor al punto que estamos mirando entonces es un REGULAR DOWN
- Viceversa es un REGULAR UP
- Si ambos puntos son mayores al punto dado y el angulo es menor a 180, entonces es un END
- Si es mayor es un MERGE
- Si ambos puntos son menores al punto dado y el angulo es menor a 180, entonces es un START
- Si el área es menor es un SPLIT



□ =start vertex

■ =end vertex

● =regular vertex

▲ =split vertex

▼ =merge vertex

Solución

Una vez creado nuestro polígono podemos comenzar el proceso de triangulación que consta de 2 pasos:

- Partir el polígono en partes monótonas
- Triángular cada polígono monótono resultante

Solución

Partir el polígono en partes monótonas

- Dependiendo el tipo de vértices calculamos el helper para cada uno de ellos
- El helper es el punto que va a ser parte de la diagonal

Solución

Caso Start Vertex

- Enviamos el id del punto de entrada al método manejar start vertex
- Como es tipo start no va a haber una nueva diagonal, entonces el helper de ese lado cuyo id es el mismo que el id del punto va a ser el mismo
- Luego se inserta en el arbol que guarda los lados con el orden de diccionario (explicar)

Solución

Caso End Vertex

- Enviamos el id del punto de entrada al método manejar end vertex
- Con un método llamado anterior, el lado anterior
- Si el helper del lado anterior es de tipo MERGE, entonces agregamos una diagonal con el id de entrada y el id de helper, que es el id a un punto
- Borramos del arbol, el lado que teníamos previamente entre nuestro punto y el anterior

Solución

Caso Split Vertex

- Enviamos el id del punto de entrada al método manejar split vertex
- Tomamos la coordenada del punto dado y luego obtenemos el lado, buscando en nuestro árbol con la condición de elegir el nodo que sea el más grande posible siendo su valor más pequeño que nuestra coordenada x
- Tomamos el helper del lado obtenido y agregamos la nueva diagonal
- Al nodo que obtuvimos en la búsqueda le seteamos el helper con el id de nuestro punto de entrada
- El helper de nuestro punto split es si mismo
- Luego se inserta en el arbol que guarda los lados con el orden de diccionario

Solución

Caso Merge Vertex

- Enviamos el id del punto de entrada al método manejar merge vertex
- Hace lo mismo que END VERTEX
- Tomamos el helper del lado obtenido y agregamos la nueva diagonal
- Al nodo que obtuvimos en la búsqueda le seteamos el helper con el id de nuestro punto de entrada
- Y además hace lo mismo que SPLIT VERTEX, pero con la condición de que si el tipo del helper obtenido del árbol es MERGE también se agrega una nueva diagonal

Solución

Caso REGULAR DOWN

- Igual que END VERTEX, pero además de borrar el lado
- Hace lo mismo que END VERTEX
- Pero, además agregamos nuestro lado al árbol, con el valor del helper seteado

Solución

Caso REGULAR UP

- Igual que SPLIT VERTEX, pero antes e insertar la nueva diagonal pregunta si el tipo del helper obtenido es MERGE

Solución

Agregar diagonal

- Se crea un lado nuevo con los dos ids dados, el tipo de los lados va a ser INSERT, así s diferencian de los input que vienen dados por el archivo
- Y se guardan en un mapa de lados, las nuevas diagonales

Solución

Partir el polígono en partes monótonas (MEJORAR ESTO)

- Eliminamos los lados que ya no necesitamos y agregamos nuevos con ls obtenidos en los pasos anteriores
- Por cada polígono monótono, insertamos los ids de los lados en una lista llamada poly
- Y luego, a todas juntas en una lista de listas de polígonos monótonos
- Iteramos esta lista y a cada polígono le aplicamos la triangulación

Solución

Triangulación (EXPLICACION ORAL)

- Tomamos la cola de prioridad que tiene nuestros puntos y una pila de puntos que se van apilando y desapilando para ir formando los triangulos, se explicará con una imagen
- Cuando se van formando los triangulos se guardan los 3 ids de los mismos en un vector de tamaño 3
- Todos los triángulos se guardan en una lista de triángulos

Solución

Teniendo los triángulos podemos pasar al segundo paso, el coloreo

- Creamos NodoGrafo, donde guardamos en cada registro el triangulo, y el centro de masa (punto), que la usaremos para el recorrido del grafo y sus vecinos, todo este registro se guarda en un arreglo de tipo NodoGrafo.
- Con el tamaño de este arreglo de NodoGrafo contruimos un Graph

Solución

Continuando

- Para cada elemento del arreglo de NodoGrafo le calculamos el vecino
- Este método pregunta si dos triángulos (mirando sus índices) comparten un lado, es decir si tienen dos indices iguales
- Una vez que se insertaron los vecinos, se agrega ese vecino al grafo con la función addEdge, los adyacentes
- Aquí aclaramos, lo que guarda la clase Graf es el id del triangulo y una lista de los ids de sus vecinos
- Una vez que tenemos el grafo creado realizamos un DFS, que nos devuelve un arreglo de posiciones y nos muestra el camino más corto a recorrer para colorear

Solución

- Ahora, dado el camino a recorrer, recorremos triangulo a triangulo y comenzamos a colorear
- Como dijimos, punto (que se encuentra en el mapa de puntos) tiene un campo color, entonces con los ids que tiene cargado el triangulo, nos dirijimos hacia los puntos
- Miramos los tres puntos, si ninguno esta coloreado nos encontramos frente al primer triangulo, que les asigna los colores aleatorios
- Si alguno está pintado se verifican las posibles casos y se colorea (caso de que uno este pintado o solo los dos)
- Como trabajamos sobre el mapa de puntos original, una vez que se coloreo el punto, en los siguientes triángulos ya van a estar pintados
- Luego recorremos el mapa de puntos que va contando la cantidad de puntos por color y nos devuelve el mínimo color

Solución

- Recorremos todo el mapa de puntos y si su color es igual que el mínimo el campo guardia se vuelve uno

Solución

- Con el mapa puntos creamos el archivo con extensión .node
- Con la lista de triángulos guardamos los tres ids de los puntos y uno propio del triagulo en un .ele
- En un archivo .color 1, 2, 3 dependiendo el color
- En un archivo .guar 0 o 1 si el punto fue guardia o no
- Estos archivos facilitaron la construcción de la interfaz