Quiz 5

Graded

Student

Paolo Vasquez Grahammer

Total Points

9.5 / 20 pts

Question 1

Winged Edge

0.5 / 6 pts

+ 6 pts **Correcto!**

+ 0 pts **Paso 1: Dividir un poligo mediante un plano**

El plano es definido como $\vec{n} \cdot (\vec{r} - \vec{P}_0) = 0$ donde $\vec{r}$ es cualquier punto en el plano. Para cada punto P_i d polígono, calculamos la distancia orientada al plano:

$$d_i = \vec{n} \cdot (\vec{P}_i - \vec{P}_0)$$

El plano intersecta al polígono si hay un cambio de signo en d_i al recorrer todos los puntos.

Para cada arista del polígono entre P_i y P_{i+1}:

$$\vec{r}(t) = \vec{P}_i + t(\vec{P}_{i+1} - \vec{P}_i)$$

donde $t \in [0, 1]$ si el punto está dentro del segmento. Podemos resolver la intersección entre la ecuación d la recta y el plano. Obtenemos:

$$t = -\frac{\vec{n} \cdot (\vec{P}_i - \vec{P}_0)}{\vec{n} \cdot (\vec{P}_{i+1} - \vec{P}_i)}$$

El punto de intersección es:

$$\vec{P}_{\text{intersección}} = \vec{P}_i + t(\vec{P}_{i+1} - \vec{P}_i)$$

Los puntos de intersección se añaden a los puntos del polígono que están en cada lado del plano para form dos nuevos polígonos.

Paso 2: Estructura Winged Edge

Definimos la clase que manejará nuestros datos:

```
class Vertex:
    self.position = position  #  (x, y, z)
    self.edge = None  # One of the edges incident

class Edge:
    self.vertex1 = vertex1  # Starting vertex
    self.vertex2 = vertex2  # Ending vertex
    self.face1 = None  # Face on the left side
    self.face2 = None  # Face on the right side
    self.next_edge1 = None  # Next edge in face1
    self.next_edge2 = None  # Next edge in face2
    self.prev_edge1 = None  # Previous edge in face1
    self.prev_edge2 = None  # Previous edge in face2

class Face:
    self.edge = None
```

Paso 3: Intersección de plano con el poliedro

Definimos la clase para el poliedro

```
class WingedEdgeStructure:
    self.vertices = []
    self.edges = []
    self.faces = []
```

Definimos la función que dividirá el poliedro en dos

```
def splittingPolyhedra(polyhedron, normalPlane, pointPlane):
    # Inicializar estructuras para los nuevos poliedros
    PositivePolyhedron = WingedEdgeStructure()
    NegativePolyhedron = WingedEdgeStructure()
    intersections = []

    # Calcular distancias orientadas y clasificar vértices
    for vertice in poliedro.vertices:
        distancia = normalPlane.dot(vertice.position - pointPlane)
        if distancia > 0:
            PositivePolyhedron.add_vertex(vertice)
        else:
            NegativePolyhedron.add_vertex(vertice)

    # Identificar aristas intersectadas y calcular puntos de intersección
    for edge in poliedro.edges:
        d1 = normalPlane.dot(edge.vertex1.position - pointPlane)
        d2 = normalPlane.dot(edge.vertex2.position - pointPlane)

        if d1 * d2 < 0:  # Arista intersecta el plano
            t = -d1 / (d2 - d1)
            P0 = edge.vertex1.position + t * (edge.vertex2.position - edge.vertex1.position)
            interseccion = Vertex(P0)
            intersections.append(interseccion)
            PositivePolyhedron.add_vertex(interseccion)
            NegativePolyhedron.add_vertex(interseccion)

    # Dividir caras intersectadas y construir nuevas caras
    for cara in poliedro.faces:
        PositiveFace = Face()
        NegativeFace = Face()
        for edge in cara.edges:
            if edge in intersections:
                PositiveFace.add_edge(edge)
                NegativeFace.add_edge(edge)
            elif edge.vertex1 in PositivePolyhedron.vertices and edge.vertex2 in PositivePolyhedron.vertices:
                PositiveFace.add_edge(edge)
            elif edge.vertex1 in NegativePolyhedron.vertices and edge.vertex2 in NegativePolyhedron.vertices
                NegativeFace.add_edge(edge)
        if PositiveFace.edges: PositivePolyhedron.add_face(PositiveFace)
        if NegativeFace.edges: NegativePolyhedron.add_face(NegativeFace)

    # Crear cara de la intersección
    newFace = Face()
    for Ei in intersections:
        newFace.add_edge(Ei)
    PositivePolyhedron.add_face(newFace)
    NegativePolyhedron.add_face(newFace)

    return PositivePolyhedron, NegativePolyhedron
```

+ 2 pts Click here to replace this description.

+ 1 pt Click here to replace this description.

✔ **+ 0.5 pts** Click here to replace this description.

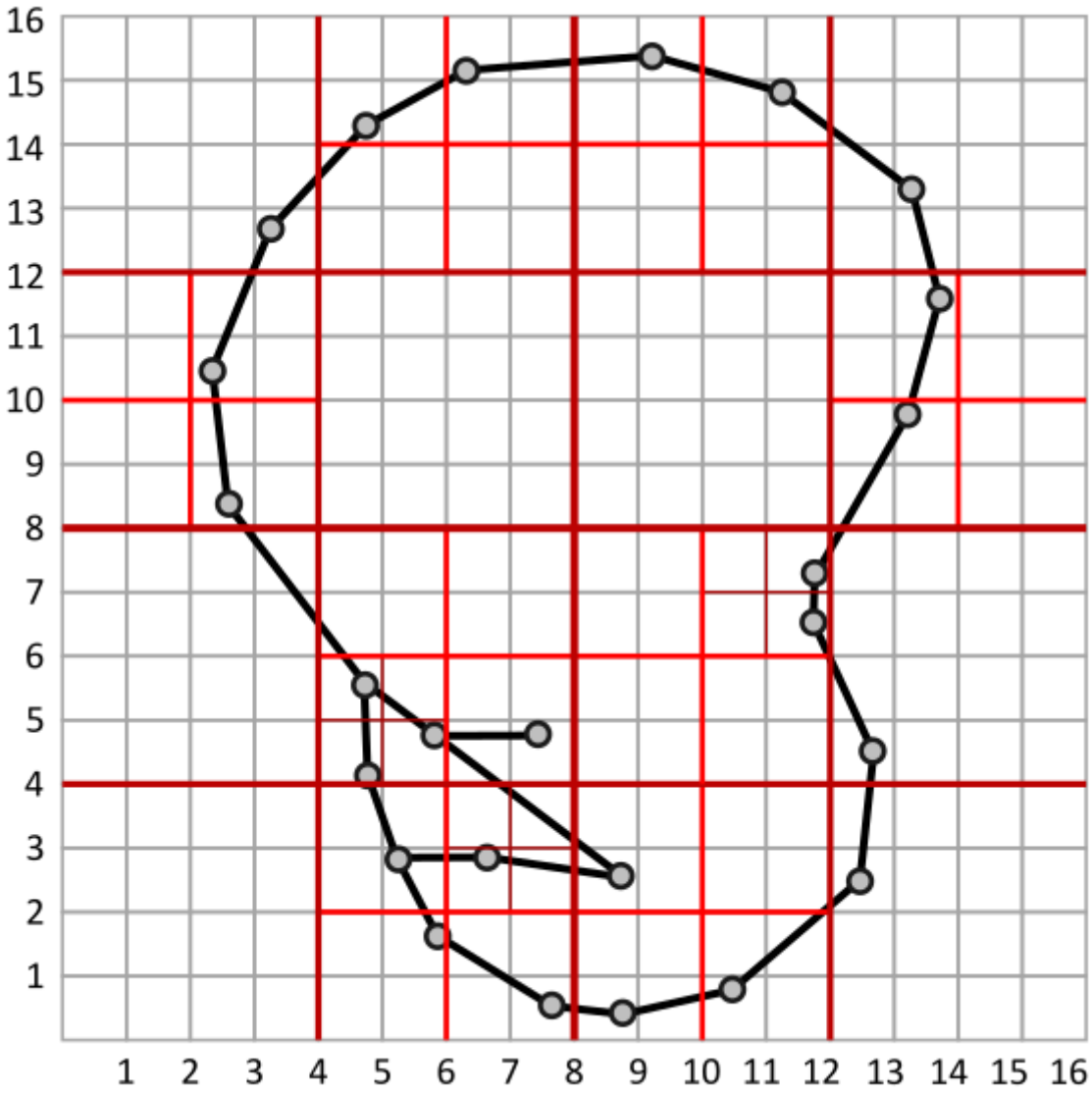
Question 2

PM2-QuadTree

4 / 4 pts

✔ **+ 4 pts** **Correcto!**

+ 0 pts



+ 3 pts Click here to replace this description.

Question 3

Hamming

1 / 5 pts

+ 5 pts **Correcto!**

+ 0 pts La distancia de Hamming entre dos cadenas de longitud n, $x = (x_1, x_2, \ldots, x_n)$ e $y = (y_1, y_2, \ldots, y_n)$, como el número de posiciones en las que los elementos correspondientes son diferentes. Formalmente:

$$d_H(x, y) = \sum_{i=1}^{n} \delta(x_i, y_i)$$

donde $\delta(x_i, y_i)$ es 1 si $x_i \neq y_i$ y 0 si $x_i = y_i$.

Demostraremos que d_H cumple las propiedades de una métrica:

a) No negatividad

$$d_H(x, y) = \sum_{i=1}^{n} \delta(x_i, y_i)$$

Dado que $\delta(x_i, y_i)$ es 0 o 1, y una suma de números no negativos es no negativa, tenemos:

$$d_H(x, y) \geq 0$$

b) Identidad

$$d_H(x, y) = 0 \iff \sum_{i=1}^{n} \delta(x_i, y_i) = 0$$

Para que la suma sea 0, cada $\delta(x_i, y_i)$ debe ser 0. Esto ocurre si y solo si $x_i = y_i$ para todo i. Por lo tanto,

$$d_H(x, y) = 0 \iff x = y$$

c) Simetría
Por la definición de δ:

$$\delta(x_i, y_i) = \delta(y_i, x_i)$$

Entonces,

$$d_H(x, y) = \sum_{i=1}^{n} \delta(x_i, y_i) = \sum_{i=1}^{n} \delta(y_i, x_i) = d_H(y, x)$$

c) Desigualdad triangular
Para $x, y, z \in \{0, 1\}^n$, tenemos que demostrar que:

$$d_H(x, z) \leq d_H(x, y) + d_H(y, z)$$

Para cada i-ésima posición, si $x_i = z_i$, entonces $\delta(x_i, z_i) = 0 \leq \delta(x_i, y_i) + \delta(y_i, z_i)$. Si $x_i \neq z_i$, enton $\delta(x_i, z_i) = 1$. En este caso, al menos uno de los términos $\delta(x_i, y_i)$ o $\delta(y_i, z_i)$ debe ser 1, lo que hace que sea al menos 1, y por lo tanto:

$$\delta(x_i, z_i) \leq \delta(x_i, y_i) + \delta(y_i, z_i)$$

Sumando sobre todas las posiciones:

$$d_H(x, z) = \sum_{i=1}^{n} \delta(x_i, z_i) \leq \sum_{i=1}^{n} (\delta(x_i, y_i) + \delta(y_i, z_i)) = \sum_{i=1}^{n} \delta(x_i, y_i) + \sum_{i=1}^{n} \delta(y_i, z_i) = d_H(x, y) +$$

+ 1.5 pts Click here to replace this description.

+ 4 pts Click here to replace this description.

+ 4.5 pts Click here to replace this description.

✔ **+ 1 pt** Click here to replace this description.

+ 2 pts Click here to replace this description.

Question 4

Line QuadTree

4 / 5 pts

+ 5 pts **Correcto!**

+ 0 pts Definimos el nodo para el Line QuadTree

```
class LineQuadtreeNode:
   def __init__(isLeaf=False, edge=None):
      self.isLeaf= isLeaf

      # [NO, NE, SO, SE]
      self.edge = edge if edge else [False, False, False, False]
      self.children = [None, None, None, None]
```

Vamos a necesitar una función que copie una sub-rama del arbol.

```
def copySubtree(node):
   if node.isLeaf:
      return LineQuadtreeNode(isLeaf=True, edge=node.edge.copy())
   result = LineQuadtreeNode(isLeaf=False)
   for i in range(4):
      if node.children[i] is not None:
         result.children[i] = copySubtree(node.children[i])
   return result
```

Utilizamos esta función para crear la función que fusiona dos nodos:

```
def fuseNodes(NodeA, NodeB):
   # Caso base
   if NodeA.isLeaf and NodeB.isLeaf:
      combinedEdge = [a or b for a, b in zip(NodeA.edge, NodeB.edge)]
      return LineQuadtreeNode(isLeaf=True, combinedEdge)

   if NodeA.isLeaf: return copySubtree(NodeB)
   if NodeB.isLeaf: return copySubtree(NodeA)

   result = LineQuadtreeNode(isLeaf=False)
   for i in range(4):
      if NodeA.children[i] is not None and NodeB.children[i] is not None:
         result.children[i] = fuseNodes(NodeA.children[i], NodeB.children[i])
      elif NodeA.children[i] is not None:
         result.children[i] = copySubtree(NodeA.children[i])
      elif NodeB.children[i] is not None:
         result.children[i] = copySubtree(NodeB.children[i])
   return result
```

Y ahora lo aplicamos para los árboles de entrada:

```
def fuseTrees(TreeA, TreeB):
   return fuseNodes(TreeA.root, TreeB.root)

L1 = LineQuadtree()
L2 = LineQuadtree()

# Fusión de los dos Line QuadTrees
LC = fuseTrees(L1, L2)
```

+ 2.5 pts Click here to replace this description.

+ 3.5 pts Click here to replace this description.

✔ **+ 4 pts** Click here to replace this description.

+ 2 pts Click here to replace this description.

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Estructura de datos Avanzados

Quiz N°5

Pregrado 2024-1

Profesor: Victor Flores Benites

Apellidos: Vasquez Grabarmer Nombres: Paolo

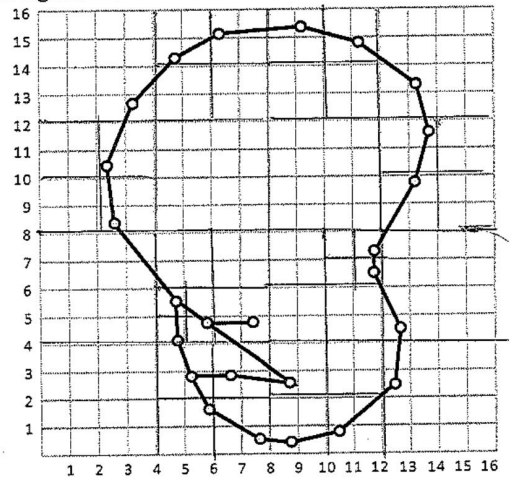
Fecha: 03/06/2024 Nota:

Indicaciones:
La Duración es de **30 minutos**.
La evaluación consta de **5 preguntas**.

1. Considere una estructura 3D en Winged Edge. Proponga un algoritmo que divida la estructura original en dos, dado un plano definido por su normal unitaria $\vec{n}$ y un punto contenido en el plano P_0. [6 pts]

(alg en hoja)

2. Muestre las particiones del PM2-QuadTree para la figura mostrada. [4 pts]



3. Demuestre que la distancia Hamming es una métrica. [5 pts]

(dem en hoja)

4. Considere dos gráficos de línea almacenados en Line QuadTrees L_1 y L_2. Proponga un algoritmo que fusione ambos gráficos. [5 pts]

(alg en hoja)