Algoritmos y Estructuras de Datos
UT7-PD5

Santiago Blanco

16-06-2025

Ejercicio #1

```
TGrafoDirigido:
---------------
tieneCiclo() -> bool
COM
  desvisitarVertices() // O(n)

  PARA CADA vert EN vertices HACER
    SI vert.visitado == FALSO ENTONCES
      resultado <- vert.tieneCiclo(nueva Lista) // O(n)
      SI resultado == VERDADERO ENTONCES
        RETORNAR VERDADERO
      FIN SI
    FIN SI
  FIN PARA CADA

  RETORNAR FALSO
FIN
```

```
TVertice:
---------------
tieneCiclo(visitados: Lista) -> bool
COM
  visitar(this)
  visitados.add(this)

  PARA CADA ady EN adyacentes HACER
    destino <- ady.getDestino()
    SI destino.visitado == FALSO ENTONCES
      SI destino.tieneCiclo(visitados) == VERDADERO ENTONCES
        RETORNAR VERDADERO
      FIN SI
    SINO SI visitados.contiene(destino) ENTONCES
      RETORNAR VERDADERO
    FIN SINO
  FIN PARA CADA

  RETORNAR FALSO
FIN
```

Tiempo de ejecución: $O(n + h)$, donde n es la cantidad de vértices y h la cantidad de aristas. Esto quiere decir que el algoritmo visita cada vértice y cada arista al menos una vez.

Ejercicio #2

```
TGrafoDirigido:
---------------
topSort() -> LinkedList<TVertice>
COM
  desvisitarVertices()
  total <- nueva LinkedList

  PARA CADA vert EN vertices HACER
    SI vert.visitado == FALSO ENTONCES
      vert.topSort(total)
    FIN SI
  FIN PARA CADA

  RETORNAR total
FIN
```

```
TVertice:
---------------
topSort(total: LinkedList) -> void
COM
  this.visitado <- VERDADERO

  PARA CADA ady EN adyacentes HACER
    destino <- ady.getDestino
    SI destino.visitado == FALSO ENTONCES
      destino.topSort(total)
    FIN SI
  FIN PARA CADA

  total.insertarPrimero(this)
FIN
```

Precondiciones

1. Debe ser un grafo dirigido válido
2. No debe tener ciclos
3. Todos los vertices se encuentran no visitados

Postcondiciones

1. Se retorna una lista enlazada que representa un orden topológico para los vértices del grafo
2. Si hay una arista $utov$, entonces u aparece antes que v en la lista.
3. Todos los vértices quedan visitados

¿Cómo haría para obtener todas las ordenaciones topológicas existentes?

Yo que sé, dejame quieto muchacho

Ejercicio #3

```
esConexo() -> boolean
COM
  vert <- vertices[0]
  recorridoDesdeVert <- bpf(vert) // O(n)

```

```
  SI recorridoDesdeVert.size() <> vertices.size() ENTONCES // O(1)
    RETORNAR FALSO
  FIN SI

  g2 <- voltearGrafo(this) // O(n)

  recorridoDesdeVert <- g2.bpf(vert) // O(n)

  SI recorridoDesdeVert.size() <> vertices.size() ENTONCES // O(1)
    RETORNAR FALSO // O(1)
  FIN SI

  RETORNAR VERDADERO // O(1)
FIN

------------------------------

voltearGrafo() -> TGrafoDirigido
COM
  graf <- Nuevo TGrafoDirigido // O(1)

  PARA CADA vert EN this.vertices HACER // O(n)
    graf.agregarVertice(vert) // O(1)
  FIN PARA CADA

  PARA CADA arista EN graf HACER // O(n)
    invertir arista // O(1)
  FIN PARA CADA

  RETORNAR graf // O(1)
FIN
```

Orden de Tiempo de ejecución: $O(n)$

Obtener componentes fuertes

```
obtenerCompFuertes(G: TGrafoDirigido) -> Lista<Lista<Vertice>>
COM
  listum <- Nueva Lista
  pilum <- Nueva Pila

  desvisitarVertices()
  PARA CADA TVertice vert EN G.vertices HACER
    SI vert.visitado == FALSE ENTONCES
      bpfPost(vert, pilum)
    FIN SI
  FIN PARA CADA

  grafoInvertido <- voltearGrafo()

  desvisitarVertices()

  compTotal <- Nueva Lista

  MIENTRAS pilum NO vacía HACER
```

```
    vert <- pilum.pop()
    SI vert en grafoInvertido NO visitado ENTONCES
      comp <- Nueva Lista
      bpfComp(vert, comp)
      compTotal.add(comp)
    FIN SI
  FIN MIENTRAS

  RETORNAR compTotal
FIN

------------------------------------------------------------

bpfPost(G : TGrafoDirigido, v : TVertice, Pila) -> void
COM
    v.visitar()
    PARA CADA u EN v.adyacentes HACER
        SI u.visitado == FALSO entonces
            bpfPost(G, u, Pila)
    FIN PARA CADA
    Pila.push(v)
FIN

------------------------------------------------------------

bpfComp(G : TGrafoDirigido, v : TVertice, componente : Lista<Vertice>) -> void
COM
    v.visitar()
    componente.add(v)

    PARA CADA u EN v.adyacentes HACER
        SI u.visitado == FALSO entonces
            bpfPost(G, u, componente)
    FIN PARA CADA
FIN

```

Tiempo de ejecución: Ejecuta una bpf ($O(n)$), luego voltearGrafo ($O(n)$), y finalmente una nueva bpf ($O(n)$) $= O(n)$