

Algoritmos y Estructuras de Datos
UT8-PD4
Santiago Blanco

27-06-2025

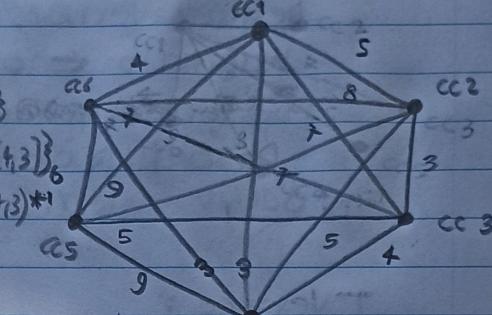
Ejercicio #1

	UT8 PD4
	Ejercicio 1
1)	TDAs necesarios: - T GrafoNoDirigido (Raíz entera) - TARISTA / TARISTAS (Flota) - TVertice (cada CC)
	El algoritmo que decidí usar fue PRIM(), porque en el PD anterior usé Kruskal
2)	PRIM(G: GrafoNoDirigido) → LISTA < TARISTA> COM O(1) O(1) costoTotal ← 0.0 aristasPrim ← Nueva LISTA O(1) O(1) SI G.vertices ES VACÍO ENTONCES RETORNAR aristasPrim O(N) etiqVertices ← obtener lista de etiquetas de G.vertices O(1) O(1) U ← Nueva LISTA de etiquetas U. agregar (etiqVertices, PRIMERO) etiqVertices. CUMPLIR PRIMERO O(N ³) O(N ²) O(1) O(1) etiqVertices. remove (ver, etiqueta) U.add (ver, etiqueta)

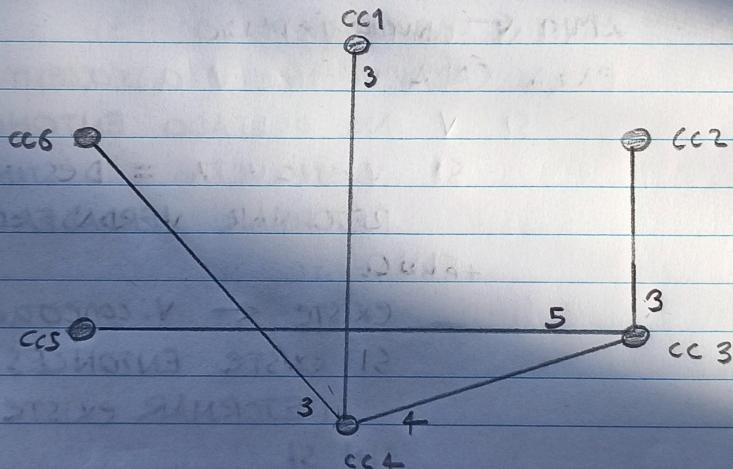
$O(1)$	AVISTAS PRIM, INSERTAR AVISTA (ari)
$O(1)$	COSTOTOTAL \leftarrow COSTOTOTAL + ari.COSTO
	FIN MIENTRAS
$O(1)$	RETORNAR AVISTAS PRIM
	FIN
	* Lo del COSTO TOTAL NO SIRVE PARA NADA
	MÉTODO AUXILIAR EN TABLISTAS
	buscarMin (verticesU : LISTA, verticesV : LISTA) \rightarrow TABLISTA
	COM
$O(1)$	tempArista \leftarrow Nueva TABLISTA
$O(1)$	taMin \leftarrow Nueva TABLISTA
$O(1)$	costoMin \leftarrow 0.0
	PARA
$O(N^2)$	PARA CADA crU EN verticesU HACER
$O(N)$	PARA CADA crV EN verticesV HACER
$O(1)$	tempArista \leftarrow buscar (crU, crV)
$O(1)$	SI tempArista \neq NULO ENTONCES
$O(1)$	SI tempArista.COSTO < COSTOMIN ENTONCES
$O(1)$	COSTOMIN \leftarrow tempArista.COSTO
$O(1)$	taMin \leftarrow tempArista
	FIN SI
	FIN SI
	FIN PARA CADA
	FIN PARA CADA
$O(1)$	RETORNAR taMin
	FIN

ORDEN TOTAL DE PRIM : $O(N^3)$

3) APIICO ALGORITMO PRIM

V-U	U	T	GRAFO ORIGINAL:
$\{2, 3, 4, 5, 6\}$	$\{1\}$	$\{\}$	
$\{2, 3, 5, 6\}$	$\{1, 4\}$	$\{(1, 4)\}$	
$\{2, 5, 6\}$	$\{1, 4, 6\}$	$\{(1, 4), (4, 6)\}$	
$\{2, 5\}$	$\{1, 3, 4, 6\}$	$\{(1, 4), (4, 6), (4, 3)\}$	
$\{5\}$	$\{1, 2, 3, 4, 6\}$	$\{(1, 4), (4, 6), (4, 3)*1$	
$\{3\}$	$\{1, 2, 3, 4, 5, 6\}$	*2	

GRAFO PRIM



*1 : ... $\{(3, 2)\}$
 *2 : $\{(1, 4), (4, 6), (4, 3), (3, 2), (3, 5)\}$

Ejercicio #2

	<u>Ejercicio 2</u>
	TGrafoNoDirigido:
	conectado (origen, destino: vertice) → booleano
	COM
O(1)	desvisitarVertices()
O(N)	resultado ← origen.conectado(destino)
	DesvisitarVertices()
	RETORNAR resultado
	FIN
	TVertice:
	conectado (destino: TVertice) → booleano
	COM
O(1)	Marcar como visitado
O(N)	PARA CADA vertice v ADYACENTE HACER
O(1)	SI v NO VISITADO ENTONCES
O(1)	SI v.etiqueta = destino ETIQUETAR ENTONCES
O(1)	RETORNAR VERDADERO
O(1)	SINO
O(N)	EXISTE ← v.conectado(destino)
O(1)	SI EXISTE ENTONCES
O(1)	RETORNAR EXISTE
O(1)	FIN SI
O(1)	FIN SI
O(1)	FIN SI
O(1)	FIN PARA CADA
O(1)	Marcar como NO VISITADO
O(1)	RETORNAR FALSO
	FIN
	TOTAL: O(N) IMPLEMENTA BPF

Lenguaje NATURAL

IMPLEMENTA BPF, donde examina cada vértice adyacente y lo compara con el destino especificado, si dentro de la recursión encuentra el destino, retorna verdadero, sino, visita todos los vértices (peor caso) y retorna falso

PRECONDICIONES

- Ningún vértice está visitado
- Es un grafo no dirigido válido

POSTCONDICIONES

- Retorna verdadero si encuentra el vértice destino durante la búsqueda
- Los vértices y el grafo no se modifican (o vuelven al estado previo)