

**ALGORITMOS Y ESTRUCTURAS DE DATOS**

**(TDSD222)**

|  |  |
| --- | --- |
|  | |
| ASIGNATURA: | Algoritmos y Estructuras de Datos |
| PROFESOR:  FECHA: | Ing. Lorena Chulde  25 – 07 - 2025 |
| PERÍODO ACADÉMICO: | 2025-A |
|  | |

**TALLER EN CLASE**

**(individual)**

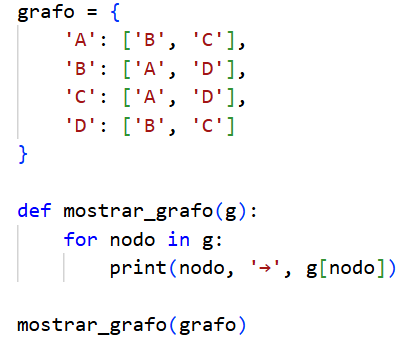
|  |
| --- |
|  |
| **TÍTULO:**  **Grafos** |
|  |
|  |
|  |
| **ESTUDIANTE**  Claudia Coello |
|  |
|  |

**OBJETIVO**

Determinar el recorrido de las estructuras de datos no lineales.

**PARTE 1: TALLER - GRAFOS**

**Grafo no dirigido y no ponderado (lista de adyacencia)**



|  |
| --- |
| grafo = {  "A" : ["B", "C"],  "B" : ["A", "D"],  "C" : ["A", "D"],  "D" : ["B", "C"]  }  def mostrar\_grafo(grafo):  for nodo in grafo:  print(nodo, " =>", grafo[nodo])  mostrar\_grafo(grafo) |
|  |

**BFS (Breadth-First Search)**

|  |
| --- |
| from collections import deque  print("Recorrido de grafos por anchura")  def bfs(grafo, inicio):  print(f"Recorrido BFS desde {inicio}: ")  visitados = set()  cola = deque([inicio])  while cola:  nodo = cola.popleft()  if nodo not in visitados:  print(nodo, end=" ")  visitados.add(nodo)  for vecino in grafo:  if vecino not in visitados and vecino not in cola:  cola.append(vecino)  print()  bfs(grafo, "A") |
|  |

**(DFS - Depth First Search)**

|  |
| --- |
| print("Recorrido de grafos profundidad")  def dfs(grafo, nodo, visitados = None):#preorden y usa pila  if visitados is None:  visitados = set()  print(nodo, end=" ")  visitados.add(nodo)  for vecino in grafo[nodo]:  if vecino not in visitados:  dfs(grafo, vecino, visitados)  dfs(grafo, "A") |
|  |

**BFS en grafo desconectado**

Si el grafo **tiene componentes desconectados**, y recorres desde cada nodo, puedes **descubrir cada componente por separado**.

|  |
| --- |
| from collections import deque  grafo = {  "A" : ["B"],  "B" : ["A"],  "C" : ["D"],  "D" : ["C"]  }  def bfsDesconectado(grafo, inicio):  visitados = set()  cola = deque([inicio])  while cola:  nodo = cola.popleft()  if nodo not in visitados:  print(nodo, end=" ")  visitados.add(nodo)  for vecino in grafo[nodo]:  if vecino not in visitados and vecino not in cola:  cola.append(vecino)  print()  print("Recorrido grafo desde cada nodo, incluso del nodo desconectado")  for nodo in grafo:  print(f"FBS desde {nodo}")  bfsDesconectado(grafo, "A") |
|  |

**Verificar si hay camino entre dos nodos:**

|  |
| --- |
| from collections import deque  grafo = {  "A" : ["B"],  "B" : ["A"],  "C" : ["D"],  "D" : ["C"]  }  def verificarCamino(grafo, origen, destino):  visitados = set()  cola = deque([origen])  while cola:  nodo = cola.popleft()  if nodo == destino:  return True  visitados.add(nodo)  for vecino in grafo[nodo]:  if vecino not in visitados and vecino not in cola:  cola.append(vecino)  return False  print("Hay camino entre A y D?", verificarCamino(grafo, "A", "D"))  print("Hay camino entre A y B?", verificarCamino(grafo, "A", "B"))  print("Hay camino entre C y D?", verificarCamino(grafo, "C", "D"))  print("Hay camino entre B y C?", verificarCamino(grafo, "B", "C")) |
|  |

**DFS para contar componentes conexas**

 Recorre el grafo con **DFS** desde cada nodo **no visitado**.

 Cada vez que comienza un nuevo DFS, significa que encontró una nueva **componente conexa**.

|  |
| --- |
| #DFS para contar componentes conexas  from collections import deque  grafo = {  "A" : ["B"],  "B" : ["A"],  "C" : ["D"],  "D" : ["C"]  }  def contarComponente(grafo):  visitados = set()  componentes = 0  def dfs(nodo):  visitados.add(nodo)  for vecino in grafo[nodo]:  if vecino not in visitados:  dfs(vecino)  for nodo in grafo:  if nodo not in visitados:  dfs(nodo)  componentes += 1  return componentes  print("Componentes conexos: ", contarComponente(grafo)) |
|  |

**Ejercicios de Dijkstra, Kruskal, TopoSort**

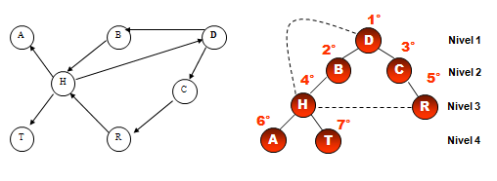
### Dijkstra – Camino mínimo

|  |
| --- |
| #Camino minimo, necesita grafo dirigido  import heapq  grafo = {  "A": [("B", 1), ("C", 4)],  "B": [("C", 2), ("D", 5)],  "C": [("D", 1)],  "D": [],  }  def djikstra(grafo, inicio):  distancias = {nodo: float("inf") for nodo in grafo}  distancias[inicio] = 0  cola = [(0,inicio)]  while cola:  distancia\_actual, nodo\_actual = heapq.heappop(cola)  for vecino, peso in grafo[nodo\_actual]:  nueva\_distancia = distancia\_actual + peso  if nueva\_distancia < distancias[vecino]:  distancias[vecino] = nueva\_distancia  heapq.heappush(cola, (nueva\_distancia, vecino))  return distancias  resultado = djikstra(grafo, "A")  print("Distancias minimas desde A: ", resultado) |
|  |

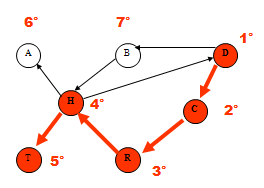
### Topological Sort (Orden topológico)

**PARTE 2: TAREA - GRAFOS**

**BFS (Breadth-First Search)**

****

**(DFS - Depth First Search)**

****

**PRESENTACIÓN:**

Al finalizar la clase, por favor entrega el taller en una hoja al profesor