UT7-TA1

✅ EJERCICIO 1

a) Matriz de adyacencias

Usamos los aeropuertos como nodos numerados del 0 al 7:

Índice	Aeropuerto
0	Artigas
1	Canelones
2	Colonia
3	Durazno
4	Florida
5	Montevideo
6	Punta del Este
7	Rocha

Matriz de adyacencias (peso de la arista o ∞ si no hay conexión):

De\A	0	1	2	3	4	5	6	7
0	∞	∞	∞	∞	∞	∞	∞	400
1	500	∞	200	170	∞	∞	90	∞
2	∞	∞	∞	∞	∞	180	∞	∞
3	∞	∞	∞	∞	∞	∞	∞	∞
4	∞	∞	∞	60	∞	∞	∞	∞
5	700	30	∞	∞	∞	∞	130	∞
6	∞	∞	∞	∞	∞	∞	∞	90
7	∞	∞	∞	∞	∞	270	∞	∞

> Nota: ∞ representa la ausencia de una arista entre dos nodos.

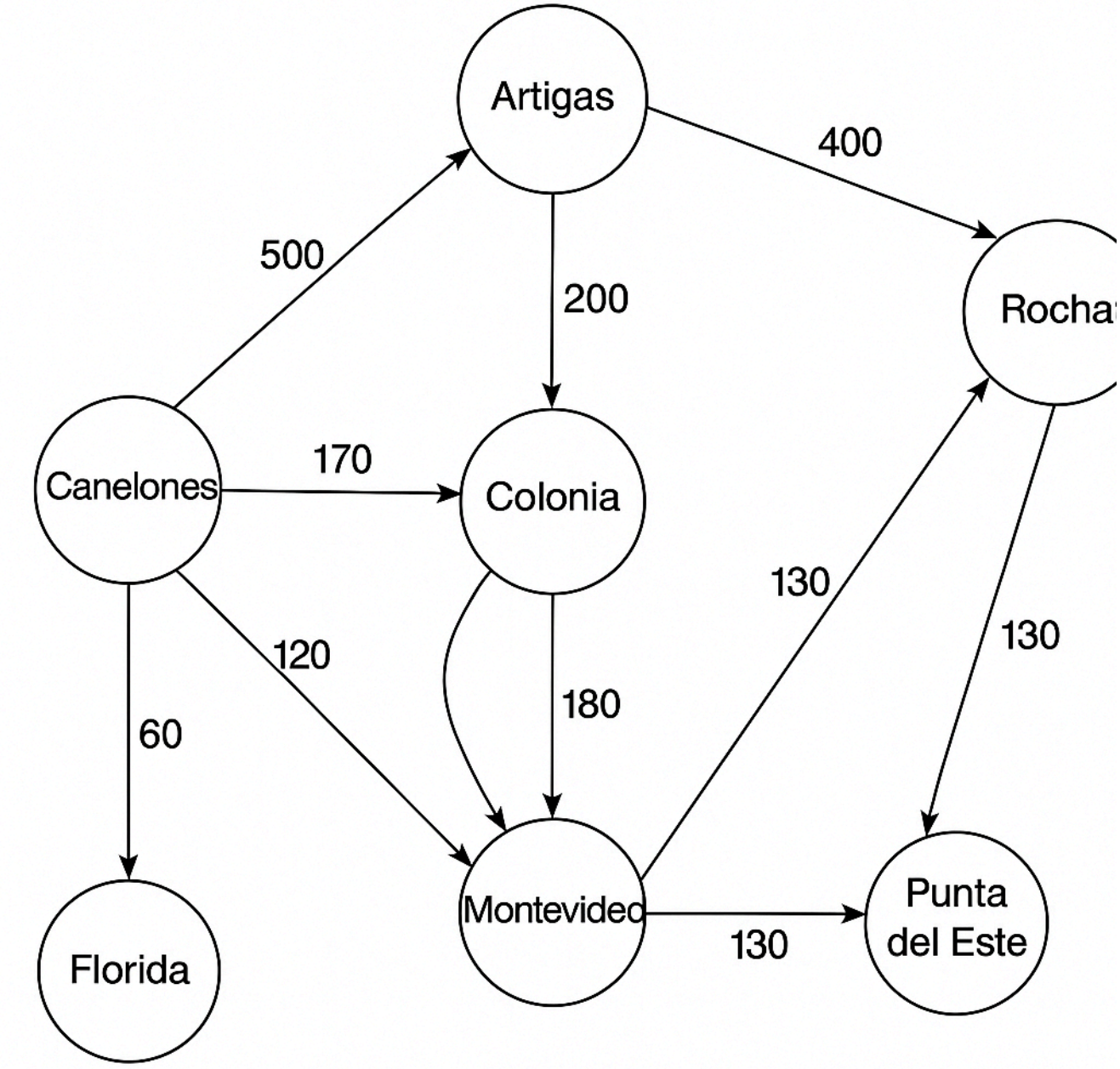
---

b) Lista de adyacencias

```
Artigas -> Rocha (400)
Canelones -> Artigas (500), Colonia (200), Durazno (170), Punta del Este
(90)
Colonia -> Montevideo (180)
Durazno -> (sin salidas)
Florida -> Durazno (60)
Montevideo -> Artigas (700), Canelones (30), Punta del Este (130)
Punta del Este -> Rocha (90)
Rocha -> Montevideo (270)
```

---

c) Representación gráfica



✅ EJERCICIO 2 - Dijkstra desde Montevideo

Paso a paso (Montevideo = nodo 5):

Paso	Nodo Actual	Distancias	Predecesores
0	Montevideo	[700, 30, ∞, ∞, ∞, 0, 130, ∞]	[5, 5, -, -, -, -, 5, -]
1	Canelones (1)	[700, 30, 230, 200, ∞, 0, 130, ∞]	[5, 5, 1, 1, -, -, 5, -]
2	Punta del Este (6)	[700, 30, 230, 200, ∞, 0, 130, 220]	[5, 5, 1, 1, -, -, 5, 6]
3	Rocha (7)	[700, 30, 230, 200, ∞, 0, 130, 220]	(ya estaban)
4	Colonia (2)	[700, 30, 230, 200, ∞, 0, 130, 220]	(ya estaba)
5	Durazno (3)	[700, 30, 230, 200, ∞, 0, 130, 220]	(ya estaba)
6	Artigas (0)	[700, 30, 230, 200, ∞, 0, 130, 220]	(ya estaba)
7	Florida (4)	[700, 30, 230, 200, 260, 0, 130, 220]	[5, 5, 1, 1, 4, -, 5, 6]

Resultado:

Destino	Distancia	Trayecto
Artigas	700	Montevideo → Artigas
Canelones	30	Montevideo → Canelones
Colonia	230	Montevideo → Canelones → Colonia
Durazno	200	Montevideo → Canelones → Durazno
Florida	260	Montevideo → Canelones → Durazno ← Florida (inverso)
Punta del Este	130	Montevideo → Punta del Este
Rocha	220	Montevideo → Punta del Este → Rocha

✅ EJERCICIO 3 - Diseño del TDA Grafo Dirigido

a) Operaciones clave del TDA Grafo

- `agregarVertice(T dato)`
- `eliminarVertice(T dato)`
- `agregarArista(T origen, T destino, int peso)`
- `eliminarArista(T origen, T destino)`
- `getAdyacentes(T vertice)`
- `dijkstra(T origen)` → `Map<T, Integer>` y/o caminos
- `tieneCiclo()` *(opcional)*
- `bfs(T origen)` y `dfs(T origen)` *(opcional)*

b) Estructuras recomendadas

- **`HashMap<T, Vertice>`**: acceso rápido a los nodos.
- Cada `Vertice` contiene un **`HashMap<T, Integer> adyacentes`** para representar las aristas salientes (destino → peso).

```
class GrafoDirigido<T> {
    Map<T, Vertice<T>> vertices;
}

class Vertice<T> {
    T dato;
    Map<T, Integer> adyacentes; // destino → peso
}
```

c) Complejidad esperada

Operación	Complejidad
Agregar vértice	O(1)
Agregar arista	O(1)
Obtener adyacentes	O(1)
Dijkstra	O((V + E) log V) usando `PriorityQueue`
Eliminar vértice/arista	O(E) o más, según implementación

---

d) Evaluación de colecciones Java

- `HashMap` → acceso O(1), ideal para nodos y listas de adyacencia.
- `TreeMap` → si se quiere orden natural de vértices (más costoso).
- `ArrayList` o `LinkedList` → para listas planas de nodos si no se necesita acceso directo.
- `PriorityQueue` → esencial para Dijkstra.