REPORTE

TÉCNICO

Milton Cruz Pánuco Castillo

1. INTRODUCCIÓN Y OBJETIVOS

Lo que logré en esta semana

- Implementé **Prim** con cola de prioridad (heap)
- Implementé **Kruskal** con Union-Find optimizado
- Desarrollé estructura **DSU** (path compression + union by rank)
- Validé con **10 tests unitarios** (10/10 pasando)
- Integré con código de Semanas 3-6
- Documenté uso de IA de forma transparente

La pregunta clave

Semana 6: ¿Cuál es la ruta más corta de A a B? (Dijkstra)
Semana 7: ¿Cuántos kilómetros MÍNIMOS de vías necesito para conectar TODAS las ciudades?

2. ARQUITECTURA DE MI IMPLEMENTACIÓN

Algoritmo de Prim

Estrategia: Construir el árbol incrementalmente desde un nodo inicial.

```python
def prim_mst(self, start_node=0):
    """
    Algoritmo de Prim para MST.
    Empieza desde start_node y va agregando aristas baratas.
    """
    visited = [False] * self.V
    pq = []

    visited[start_node] = True
    for neighbor, weight in self.adj[start_node]:
        heapq.heappush(pq, (weight, start_node, neighbor))

    mst_edges = []
    mst_cost = 0

    while pq:
        weight, u, v = heapq.heappop(pq)

        if visited[v]:
            continue

        visited[v] = True
        mst_edges.append((u, v, weight))
        mst_cost += weight

        for next_node, next_weight in self.adj[v]:
            if not visited[next_node]:
                heapq.heappush(pq, (next_weight, v, next_node))

    return mst_edges, mst_cost
```

Complejidad: O(E log V) ≈ 20 operaciones para mi red

Algoritmo de Kruskal

Estrategia: Ordenar aristas y seleccionar las que no crean ciclos.

```python
def kruskal_mst(self):
    """
    Algoritmo de Kruskal para MST.
    Ordena aristas por peso y va agregando las que no crean ciclos.
    """
    mst_cost = 0
    mst_edges = []
    dsu = self.DSU(self.V)

    sorted_edges = sorted(self.edges, key=lambda item: item[2])

    for u, v, w in sorted_edges:
        if dsu.union(u, v):
            mst_edges.append((u, v, w))
            mst_cost += w

    return mst_edges, mst_cost
```

Complejidad: O(E log E) ≈ 20 operaciones para mi red

Union-Find (DSU) - La clave de Kruskal

Tomé ayuda de IA para implementar esto correctamente.

```python
def __init__(self, n):
    self.parent = list(range(n))
    self.rank = [0] * n

def find(self, i):
    """Encuentra la raiz del conjunto que contiene a i"""
    if self.parent[i] != i:
        self.parent[i] = self.find(self.parent[i])
    return self.parent[i]

def union(self, i, j):
    """Une los conjuntos que contienen a i y j"""
    root_i = self.find(i)
    root_j = self.find(j)

    if root_i != root_j:
        if self.rank[root_i] < self.rank[root_j]:
            self.parent[root_i] = root_j
        elif self.rank[root_i] > self.rank[root_j]:
            self.parent[root_j] = root_i
        else:
            self.parent[root_j] = root_i
            self.rank[root_i] += 1
        return True
    return False
```

3. RESULTADOS DE MIS TESTS - 10/10 PASANDO

#	Test	Qué probé	Resultado
1	test_prim_simple	Grafo básico 4 nodos	PASS
2	test_kruskal_simple	Mismo grafo con Kruskal	PASS
3	test_prim_triangulo	Caso minimal 3 nodos	PASS
4	test_kruskal_triangulo	Triángulo con Kruskal	PASS
5	test_mismo_resultado	Prim = Kruskal (costo)	PASS
6	test_union_find_basico	Union-Find correcto	PASS
7	test_union_mismo_conjunto	Detecta ciclos	PASS
8	test_path_compression	Optimización DSU*	PASS
9	test_grafo_dos_nodos	Caso edge minimal	PASS
10	test_numero_aristas	MST tiene V-1 aristas	PASS

Test #8: Tomé ayuda de IA para diseñar este test

4. ANÁLISIS DE MI RED URBANA

Mi red actual (5 ciudades de Semana 6)

Conexiones existentes:

- Centro ↔ B: 50.5 km
- Centro ↔ C: 80.0 km
- Centro ↔ D: 95.0 km
- B ↔ D: 30.0 km
- C ↔ D: 45.5 km
- C ↔ E: 70.0 km
- D ↔ E: 25.0 km

Total: 391.5 km de vías

Resultado del MST (Prim y Kruskal)

Aristas seleccionadas:

1. D-E: 25.0 km
2. B-D: 30.0 km
3. D-C: 45.5 km

4. Centro-B: 50.5 km

Total MST: 151.0 km

Hallazgo crítico: Ahorro de 240.5 km (61.4%)

Métrica	Red Actual	MST	Ahorro
Kilómetros	391.5 km	151.0 km	**240.5 km (61%)**
Aristas	7	4	3 redundantes
Conectividad	Completa	Completa	Igual

Aristas eliminadas por redundantes:

- Centro-C (80 km)
- Centro-D (95 km)
- C-E (70 km)

5. COMPARACIÓN PRIM VS KRUSKAL

Aspecto	Prim	Kruskal
Estrategia	Crece desde un nodo	Une componentes
Estructura	Cola de prioridad	Union-Find
Complejidad	O(E log V)	O(E log E)
Grafo denso	Mejor	Más lento
Grafo disperso	Igual	Mejor
Implementación	Moderada	Difícil (DSU)

Para mi red de 5 ciudades: Ambos son igualmente rápidos (~0.01 ms)

6. CONEXIÓN CON SEMANA 6

MST vs Dijkstra

Aspecto	Dijkstra (Semana 6)	MST (Semana 7)
Pregunta	¿Ruta más corta A→B?	¿Mínimo km total?
Output	Distancias individuales	1 árbol global
Uso	GPS, navegación	Diseño de redes

Ejemplo: Centro → E

- **Dijkstra:** 105.5 km (vía B→D→E)
- **MST:** Mismo camino en el árbol (¡casualidad!)

Diferencia clave: MST minimiza costo TOTAL, no rutas individuales.

7. CASOS DE USO

Caso 1: Red eléctrica desde cero

Costo: $1 millón por km

Opción	Kilómetros	Costo
Red completa	391.5 km	$391.5M
MST	151.0 km	$151.0M
Ahorro	240.5 km	**$240.5M (61%)**

Caso 2: Construcción por fases

Orden recomendado (según Kruskal):

1. Año 1: D-E (25 km) - Más barata
2. Año 2: B-D (30 km) - Conecta 3 ciudades
3. Año 3: D-C (45.5 km) - Agrega C
4. Año 4: Centro-B (50.5 km) - Completa la red

Total: 4 años, $151M distribuidos

8. REFLEXIÓN SOBRE USO DE IA

Dónde pedí ayuda

1. Implementación de Union-Find:

 # Comentario en mi código:
 # En esta parte tome ayuda de la IA para implementar Union-Find correctamente

- Pedí: Explicación de path compression y union by rank
- Por qué: Optimizaciones no triviales
- Validé: Con tests propios

2. Tests de path compression:

 # En test_path_compression
 # En esta parte tome ayuda de la IA para diseñar tests de path compression

Qué hice YO

- Toda la lógica de Prim (100% propia)

- Lógica principal de Kruskal (solo DSU con ayuda)
- 8 de 10 tests sin ayuda
- Todo el análisis de red urbana
- Este reporte completo

9. CONCLUSIONES

Lo que funcionó

1. Ambos algoritmos correctos (10/10 tests)
2. Integración perfecta con Semanas 3-6
3. Descubrí ahorro de 61% en kilómetros
4. Entiendo profundamente Union-Find

Hallazgos en mi red

- **MST necesita solo 151 km** vs 391.5 km actuales
- **Arista Centro-D es inútil** (nunca se usa)
- **B-D es crítica** (punto único de fallo)

Aprendizaje principal

MST ≠ Rutas más cortas

- MST: Minimiza costo TOTAL de conectividad
- Dijkstra: Minimiza rutas INDIVIDUALES
- Son COMPLEMENTARIOS, no excluyentes