Reporte Semanal

Ramirez Moreno
Mauricio Damian

1. INTRODUCCIÓN Y OBJETIVOS

Logros de esta semana

- Implementé el algoritmo de Prim utilizando una cola de prioridad (heap).
- Implementé Kruskal apoyándome en una estructura Union-Find optimizada.
- Construí la estructura DSU con *path compression* y *union by rank*.
- Validé la solución con 10 pruebas unitarias (todas aprobadas: 10/10).
- Integré correctamente el trabajo con el código construido entre las semanas 3 y 6.
- Documenté de forma transparente en qué partes utilicé apoyo de IA.

Pregunta clave

- Semana 6: ¿Cuál es la ruta más corta entre dos puntos A y B? (Dijkstra)
- Semana 7: ¿Cuál es la cantidad mínima de kilómetros que necesito para conectar TODAS las ciudades? (MST)

2. ARQUITECTURA DE LA IMPLEMENTACIÓN

Algoritmo de Prim

```python
def prim_mst(self, start_node=0):

    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
```

Idea principal: construir el árbol mínimo expandiéndolo desde un nodo inicial.

Complejidad: O(E log V) → para mi red equivale a unas ~20 operaciones.

Algoritmo de Kruskal

```python
def kruskal_mst(self):

    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
```

Estrategia: ordenar las aristas y elegir solo las que no generen ciclos.

Complejidad: O(E log E) → también cerca de ~20 operaciones en mi red.

Union-Find (DSU)

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

def find(self, i):

    if self.parent[i] != i:
        self.parent[i] = self.find(self.parent[i])
    return self.parent[i]

def union(self, i, 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
```

Es la base para que Kruskal funcione correctamente.

Pedí apoyo a IA para implementarlo con las optimizaciones adecuadas.

3. RESULTADOS DE LAS PRUEBAS (10/10 EXITOSAS)

Tabla de tests

Test	Qué verifica	Resultado
test_prim_simple	Grafo básico de 4 nodos	PASS
test_kruskal_simple	Mismo grafo evaluado con Kruskal	PASS
test_prim_triangulo	Caso mínimo de 3 nodos	PASS
test_kruskal_triangulo	Triángulo con Kruskal	PASS
test_mismo_resultado	Que Prim y Kruskal tengan el mismo costo	PASS
test_union_find_basico	Funcionamiento correcto del DSU	PASS
test_union_mismo_conjunto	Detección de ciclos	PASS
test_path_compression	Optimización del DSU (*con ayuda de IA*)	PASS
test_grafo_dos_nodos	Caso con 2 nodos	PASS
test_numero_aristas	Que el MST tenga V–1 aristas	PASS

4. ANÁLISIS DE LA RED URBANA

Conectividad actual (Semana 6)

Conexión	Distancia (km)
Centro – B	50.5
Centro – C	80.0
Centro – D	95.0
B – D	30.0
C – D	45.5
C – E	70.0
D – E	25.0
Total	391.5 km

Aristas seleccionadas por el MST (Prim y Kruskal)

Arista	Distancia (km)
D – E	25.0
B – D	30.0
D – C	45.5
Centro – B	50.5
	151.0 km

Comparación general

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

Aristas redundantes:

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

5. COMPARACIÓN ENTRE PRIM Y KRUSKAL

Aspecto	Prim	Kruskal
Estrategia	Crece desde un nodo	Une varios componentes
Estructura	Cola de prioridad	Union-Find
Complejidad	O(E log V) (mejor en grafos densos)	O(E log E) (eficiente en grafos dispersos)
Implementación	Más sencilla	Más técnica (DSU)
Velocidad en mi red	~0.01 ms	~0.01 ms

6. RELACIÓN CON LA SEMANA 6

Aspecto	Dijkstra	MST
Pregunta que responde	¿Cuál es la ruta más corta entre A y B?	¿Cuál es el mínimo total de kilómetros?
Resultado	Distancia individual	Un árbol completo
Uso típico	Navegación y GPS	Diseño de infraestructura
Ejemplo Centro → E	105.5 km	Usa el mismo camino (por coincidencia)

Idea clave:
El MST reduce el costo TOTAL, pero no optimiza rutas individuales.

7. CASOS DE USO

Caso 1: Construcción de red desde cero

Costo estimado: $1 millón por km

Opción	Kilómetros	Costo	Ahorro
Red completa	391.5 km	$391.5M	—
MST	151.0 km	$151.0M	$240.5M

Caso 2: Construcción en etapas (orden recomendado)

Año	Arista	Distancia (km)
1	D – E	25
2	B – D	30
3	D – C	45.5
4	Centro – B	50.5

8. USO DE IA EN MI PROCESO

Dónde pedí apoyo

Union-Find:

- Solicité explicación de *path compression* y *union by rank*.
- Razón: son optimizaciones avanzadas.
- Validación: añadí pruebas propias.

Prueba de path compression:

- Recibí ayuda para diseñar el test.

Qué desarrollé personalmente

- Implementación completa del algoritmo de Prim.
- Lógica central de Kruskal (excepto DSU).
- 8 de los 10 tests.
- Todo el estudio de la red urbana.
-

9. CONCLUSIONES

- Los algoritmos funcionan correctamente (10/10 pruebas).
- La integración con el trabajo previo fue exitosa.
- El MST identificó un ahorro del 61% en kilómetros.
- Comprendí a profundidad el funcionamiento de Union-Find.

Hallazgos en mi red

- El MST requiere solo 151 km, contra 391.5 km actuales.
- La conexión Centro–D es completamente innecesaria.
- La arista B–D es un punto crítico de la red.

Aprendizaje principal

- MST no sirve para rutas más cortas.
- MST: minimiza el costo total de conexión.

Dijkstra: minimiza distancias entre pares de nodos.
Ambos se complementan.