
# Analisis de Algoritmos 2024-1

## Tarea 09: Prim, Kruskal y Dijkstra

**Profesor:** Profesor(a): María de Luz Gasca Soto
**Ayudantes:** Rodrigo Fernando Velázquez Cruz
Teresa Becerril Torres
**Alumno:** Alvaro Ramírez López **N° cuenta:** 316276355


1. Proporcionar una gráfica conexa $G = (V, A)$ con al menos 19 vértices y al menos 43 aristas con pesos positivos en el intervalo $[1, 9] \subset Z$; deberá haber al menos cuatro aristas de cada costo $c$, $c \in [1, 9] \subset Z$. Aplicar los siguientes algoritmos a la gráfica dada G, ilustrando cómo van transformandose las estructuras y mostrando al final los valores de las etiquetas para cada vértice.
   a) Prim usando Heaps Binarios.
   b) Kruskal usando Conjuntos Ajenos con union por tamaño.
   c) Dijkstra usando Colas Binomiales. Modificar la gráfica $G = (V, A)$ dada: seleccionar un vértice como fuente, $s$, y dar dirección a las aristas para aplicar Dijsktra.
2. Sea a una arista de peso mínimo de una gráfica $G = (V, A)$ con pesos en las aristas.
   a) Modificar tanto el Algoritmo Prim como el Kruskal para que la arista a siempre aparezca en el árbol generador de peso mínimo.
   b) Calcular el desempeño computacional de los algoritmos propuestos, indicando las estructuras de datos usadas para lograr tal desempeño.

**Solución:**

Para lograr que una arista específica, denotada como 'a', siempre aparezca en el árbol generador de peso mínimo, se pueden realizar modificaciones en los algoritmos de Prim y Kruskal. A continuación, se presentan las modificaciones para ambos algoritmos:

**Modificación del Algoritmo de Prim:**

1. **Inicialización:** Inicializa el árbol generador mínimo con la arista 'a'.
2. **Selección de la Arista:** En cada paso, selecciona la arista de peso mínimo que conecta un vértice en el árbol actual con un vértice fuera del árbol, garantizando que la arista 'a' siempre sea seleccionada si es posible.
3. **Actualización de Conjuntos:** Actualiza los conjuntos de vértices que forman parte del árbol y los que están fuera del árbol.
4. **Repetición:** Repite el proceso hasta que todos los vértices estén en el árbol generador mínimo.

**Modificación del Algoritmo de Kruskal:**

1. **Orden de las Aristas:** Antes de comenzar el algoritmo, asegúrate de que la arista 'a' esté incluida en la lista de aristas y se ordene primero.
2. **Unión de Conjuntos:** Durante la ejecución del algoritmo, al unir dos conjuntos de vértices, verifica que la arista 'a' esté incluida en el árbol generador mínimo.

3. **Repetición:** Continúa con el proceso hasta que todos los vértices estén en un solo conjunto.

**Desempeño Computacional:**

El desempeño computacional de ambos algoritmos dependerá en gran medida de las estructuras de datos utilizadas. Aquí hay algunas consideraciones generales:

**Algoritmo de Prim modificado:**

- **Estructura de Datos para Mantener las Aristas:** Puedes usar una cola de prioridad (heap binario o Fibonacci) para mantener las aristas ordenadas por peso.
- **Estructura de Datos para Mantener Conjuntos de Vértices:** Puedes usar una estructura de conjunto disjunto (Union-Find) para realizar operaciones de unión y encontrar eficientemente.

**Algoritmo de Kruskal modificado:**

**Estructura de Datos para Mantener las Aristas:** Al igual que en Prim, una cola de prioridad es útil para mantener las aristas ordenadas por peso.

**Estructura de Datos para Mantener Conjuntos de Vértices:** También puedes usar una estructura de conjunto disjunto (Union-Find) para realizar las operaciones de unión y encontrar de manera eficiente.

En términos de complejidad temporal, tanto Prim como Kruskal tendrán una complejidad de $O(E \log V)$, donde $E$ es el número de aristas y $V$ es el número de vértices. Las operaciones de conjuntos disjuntos son cruciales para lograr eficiencia en ambos algoritmos.

Es importante señalar que la complejidad exacta dependerá de la implementación específica y de la elección de las estructuras de datos.

3. Proyecto $\Gamma$: Sea $D = (V, A, \delta)$ una gráfica disconexa con pesos sobre las aristas, con $|V| = n$, $|A| = m$ y $\delta : A \to R+$.

   a) Diseñar un algoritmo que determine el bosque generador de peso mínimo para D, el algoritmo debe tener complejidad $O(m \log m)$, en el peor de los casos. El algoritmo debe ser capaz de mostrar los árboles obtenidos y los correspondientes pesos.

   b) Indicar las Estructuras de Datos con las que se alcanza el tiempo pedido y justificar formalmente el desempeño computacional del algoritmo propuesto.

   c) Proporcionar una gráfica D con al menos 27 vértices, al menos 39 aristas con pesos positivos y al menos cuatro componentes conexas. Aplicar detalladamente el algoritmo propuesto en el Ejercicio 3 a la gráfica disconexa D creada.

**Solución:**

Aqui esta la solucion del problema 3.

4. **Opcional.** Sea $G = (V, A)$ una gráfica conexa con pesos positivos sobre las aristas. Supongamos que el costo de un árbol generador se define como el producto de los costos en las aristas

   a) Diseñar un algoritmo que determine el árbol generador de peso máximo, usando tal regla.

   b) Calcular el desempeño computacional del algoritmo propuesto, indicando las estructuras de datos usadas para lograr tal desempeño.

**Solución:**

**a)Diseñar un algoritmo que determine el árbol generador de peso máximo, usando tal regla.**

Para resolver este problema, puedes utilizar el algoritmo de Kruskal modificado para maximizar el peso del árbol generador. Aquí te doy una descripción del algoritmo y luego abordo la cuestión del desempeño computacional y las estructuras de datos utilizadas.

Algoritmo para encontrar el árbol generador de peso máximo:

1. **Ordenar las aristas:** Ordena todas las aristas en orden no decreciente de peso.
2. **Inicializar el árbol generador:** Inicializa un árbol generador vacío.
3. **Agregar aristas de mayor a menor:** Recorre las aristas ordenadas de mayor a menor peso y agrega cada arista al árbol generador si no forma un ciclo con las aristas ya seleccionadas.

Este enfoque modificado del algoritmo de Kruskal seleccionará aristas de mayor peso en lugar de menor peso, asegurando así que el árbol generador resultante tenga el peso máximo.

**b) Calcular el desempeño computacional del algoritmo propuesto, indicando las estructuras de datos usadas para lograr tal desempeño.**

**Ordenar las aristas:** Este paso requiere tiempo $(O(E \log E))$, donde $(E)$ es el número de aristas.

**Estructuras de datos:** Para mantener el seguimiento de los conjuntos disjuntos y verificar si agregar una arista forma un ciclo, puedes utilizar una estructura de datos eficiente como una estructura de conjuntos disjuntos (Disjoint Set Union, DSU). La operación de unión en DSU puede realizarse en $(O(\alpha(V)))$ en promedio, donde $(\alpha(V))$ es la inversa de la función de Ackermann, y $(V)$ es el número de vértices.

**Recorrer las aristas ordenadas:** Este paso tiene un tiempo de ejecución de $(O(E))$, ya que se recorren todas las aristas una vez.

Por lo tanto, el tiempo total de ejecución es dominado por el costo de ordenar las aristas, lo que resulta en $(O(E \log E))$ en el peor de los casos.

El espacio adicional requerido para mantener la estructura de conjuntos disjuntos y otras variables auxiliares es $(O(V + E))$.

En resumen, el desempeño computacional del algoritmo propuesto es $(O(E \log E))$, y las estructuras de datos utilizadas son principalmente la estructura de conjuntos disjuntos y un arreglo para mantener las aristas ordenadas.