Tarea 9

Juárez Torres Carlos Alberto

May 25, 2023

1 Ejercicio 1

2 Sea a una arista de peso mínimo de una gráfica $G = (V, A)$ con pesos en las aristas.

2.1 Modificar tanto el Algoritmo Prim como el Kruskal para que la arista a siempre aparezca en el árbol generador de peso mínimo.

- **Modificación del algoritmo de Prim:**
  Para garantizar que la arista a siempre aparezca en el árbol generador de peso mínimo, podemos modificar el algoritmo de Prim de la siguiente manera:

  1. Inicializar un conjunto vacío MST para almacenar el árbol generador de peso mínimo.
  2. Inicializar una cola de prioridad Q para almacenar los vértices y sus costos de conexión.
  3. Insertar el vértice inicial arbitrario en Q con costo 0.
  4. Mientras Q no esté vacío:
     (a) Extraer el vértice v con el costo mínimo de Q.
     (b) Si el vértice v está en MST, ignorarlo y continuar al siguiente vértice en Q.
     (c) Agregar la arista que conecta v con su vértice padre en MST al conjunto MST.
     (d) Insertar todos los vecinos no visitados de v en Q con el costo correspondiente.
  5. Devolver MST.

  La modificación consiste en agregar una condición en el paso 4b para verificar si la arista que conecta el vértice v con su vértice padre en el árbol generador de peso mínimo es la arista a. Si es así, se agrega al conjunto MST sin verificar si el vértice v está en MST.

- **Modificación del algoritmo de Kruskal:**
  Para asegurar que la arista a siempre esté presente en el árbol generador de peso mínimo, podemos modificar el algoritmo de Kruskal de la siguiente manera:

  1. Ordenar todas las aristas del grafo en orden no decreciente de peso.
  2. Inicializar un conjunto vacío MST para almacenar el árbol generador de peso mínimo.
  3. Para cada arista (u, v) en el orden de las aristas ordenadas:
     (a) Si la arista (u, v) es la arista a, agregarla al conjunto MST.
     (b) Si agregar la arista (u, v) a MST no crea un ciclo, agregarla a MST.
  4. Devolver MST.

  La modificación consiste en agregar una condición en el paso 3a para verificar si la arista actual es la arista a. Si es así, se agrega directamente al conjunto MST sin realizar ninguna verificación adicional.

2.2 Calcular el desempeño computacional de los algoritmos propuestos

1. **Modificación del algoritmo de Prim:**
   La modificación del algoritmo de Prim tiene un bucle principal que se ejecuta hasta que la cola de prioridad Q esté vacía. En cada iteración, se extrae el vértice con el costo mínimo de Q y se realizan operaciones constantes para verificar y agregar aristas al conjunto MST.

La cola de prioridad Q puede implementarse con una estructura de datos eficiente, como un montículo binario o un montículo de Fibonacci, lo que permite una inserción y extracción eficiente de elementos en tiempo logarítmico. Suponiendo que el tamaño del grafo es V (número de vértices) y el número total de aristas es E, el tiempo de ejecución del algoritmo modificado de Prim es aproximadamente $O((V + E) \log V)$.

2. **Modificación del algoritmo de Kruskal:**
   La modificación del algoritmo de Kruskal también tiene un bucle principal que itera sobre todas las aristas ordenadas. En cada iteración, se realiza una verificación constante para determinar si la arista actual es la arista a y si agregarla crea un ciclo en el árbol generador.

   El ordenamiento de las aristas se realiza previamente y tiene un tiempo de ejecución de $O(E \log E)$ si se utiliza un algoritmo de ordenamiento eficiente, como el ordenamiento rápido (quicksort) o el ordenamiento por mezcla (merge sort). Luego, el bucle principal recorre todas las aristas en $O(E)$ y las operaciones constantes de verificación y agregado de aristas al conjunto MST también se ejecutan en tiempo constante. En total, el tiempo de ejecución del algoritmo modificado de Kruskal es aproximadamente $O(E \log E)$.

En términos generales, la complejidad asintótica del algoritmo modificado de Prim es mejor que la del algoritmo modificado de Kruskal, ya que E puede ser mayor o igual que V, y en el peor de los casos, E puede ser del orden de V^2. Sin embargo, es importante tener en cuenta que la eficiencia real de los algoritmos puede depender de factores adicionales, como la implementación específica, las estructuras de datos utilizadas y las características del grafo en particular.

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

3.1 Diseñar un algoritmo que determine el árbol generador de peso máximo, usando tal regla

La modificación consiste en ordenar las aristas en orden decreciente en lugar de creciente.

1. Ordenar todas las aristas del grafo en orden decreciente de peso.
2. Inicializar un conjunto vacío MST para almacenar el árbol generador de peso máximo.
3. Para cada arista (u, v) en el orden de las aristas ordenadas:
   (a) Si agregar la arista (u, v) a MST no crea un ciclo, agregarla a MST.
4. Devolver MST.

La modificación clave es el paso 1, donde las aristas se ordenan en orden decreciente en lugar de creciente. Esto asegura que se seleccionen primero las aristas más pesadas durante el proceso de construcción del árbol generador.

3.2 Calcular el desempeño computacional del algoritmo propuesto, indicando las estructuras de datos usados para lograr tal desempeño.

El desempeño computacional del algoritmo propuesto es similar al del algoritmo de Kruskal estándar, ya que implica ordenar las aristas y realizar operaciones de conjuntos disjuntos. A continuación se describen las estructuras de datos utilizadas y el desempeño esperado:

- **Conjunto de vértices y aristas:** Se pueden utilizar estructuras de datos como matrices, listas de adyacencia o listas de aristas para almacenar los vértices y las aristas.
- **Estructura de datos de conjuntos disjuntos:** Para verificar si agregar una arista crea un ciclo, se requiere una estructura de datos eficiente para mantener y fusionar conjuntos disjuntos. La estructura de datos de Union-Find es comúnmente utilizada y tiene un desempeño de tiempo casi constante para las operaciones de unión y búsqueda de conjuntos.

El desempeño computacional del algoritmo propuesto es O(—E— log —V—), donde —V— es el número de vértices y —E— es el número de aristas en el grafo. Esto se debe principalmente al tiempo requerido para ordenar las aristas en orden decreciente. La estructura de datos de conjuntos disjuntos (Union-Find) también contribuye a la complejidad del algoritmo, pero su impacto es menor en comparación con el tiempo de ordenación de las aristas.