TALLER N°2

Minumum Spanning Tree

usach

Taller de programación 2-2023

Fecha: 06 de noviembre 2023

Autor: Jaime Riquelme Olguin

TALLER N°2

Minumum Spanning Tree

Explicación breve del algoritmo

El algoritmo realizado para el Minimum Spanning Tree fue basado en el algoritmo de Kruskal. Básicamente, obtenemos la matriz de adyacencia con sus pesos del archivo entregado, solamente leemos la mitad de ella ya que es una matriz simétrica. A medida que obtenemos cada línea del archivo, vamos creando los nodos y aristas respectivas con sus pesos. Una vez que tenemos todas las aristas, procedemos a utilizar el algoritmo de Kruskal para encontrar el MST. En cada iteración, Kruskal obtiene el menor de los pesos existentes entre los vértices, verifica que al seleccionar ese vértice no se generen ciclos. Si este no genera, une el nodo de inicio con el final en un mismo conjunto para así ir, posteriormente, verificando que no se formen ciclos.

Heurísticas o técnicas utilizadas

Se utilizaron diversas técnicas para resolver el MST. La principal de ellas fue el algoritmo de Kruskal, que es conocido por ser un método Greedy. (S., 2009, pág. 652; S., 2009) Esta técnica se centra en hacer selecciones óptimas a nivel local para construir el árbol. Además, se empleó una cola de prioridad para el manejo de las aristas, permitiendo de esta manera acceder siempre a la arista de menor costo. Finalmente, se utilizó una tabla hash para el conjunto de nodos, facilitando así la verificación rápida para evitar la formación de ciclos en el árbol.

Funcionamiento del programa

El problema se representa mediante objetos tipo aristas, que cada arista contiene un nodo inicial, final y el coste para llegar de uno al otro.

Se define un objeto arista como:

- Nodo inicial
- Nodo final
- Coste

Para gestionar las aristas, como se mencionó anteriormente se utilizó para la solución el algoritmo de kruskal, por lo tanto, se emplea un Heap o cola de prioridad para el manejo de las aristas, así al extraer un elemento(pop), se obtiene siempre la arista con menor coste posible.

Además, se utiliza una tabla hash para la contención de los nodos, y posterior verificación de bucles de manera eficiente.

La función “Kruskal”, pertenece a la clase grafo, que se encarga de resolver y obtener el MSP del archivo ingresado por consola.Luego de leer el archivo ingresado, crear y guardar cada vertice dentro del Heap, y para cada nodo crear un conjunto con si mismo, se procede a hacer llamada a la función para resolver el problema.

Dentro de la función:

se inicializan distintas variables como, pesototal, NumeroAristas, NumeroAristasRevisadas. Y se inicia el bucle para encontrar el MST respectivo.
Dentro del bucle: (Mientras el heap de aristas sea distinto de vacío o Numero de aristas ingresadas sea < numero de nodos - 1)

- Aumentamos en 1 el número de aristas revisadas
- Realizamos Pop para obtener la arista de menor coste.
- Buscamos el set o padre al que pertenece el origen y destino.
- Si el set o padre de origen y destino es distinto entre sí: (No forma ciclos)
  - Aumentamos el número de aristas.
  - Unimos al mismo set o padre el origen con el destino.
  - Mostramos por consola la arista.
  - Aumentamos el peso total, sumando el coste de la arista verificada.

Finalmente Ingresamos el número de aristas revisadas al grafo y mostramos el peso total obtenido.

Aspectos de implementación y eficiencia

La eficiencia de una solución no solo radica en el algoritmo empleado, sino también en la forma en que se implementa y las estructuras de datos que se utilizan. En el caso de la solución propuesta para el problema del Árbol de Expansión Mínima (MST), se pueden destacar varios factores que contribuyen a su eficiencia:

1. **Uso del algoritmo Kruskal**: Este algoritmo es conocido por su eficiencia en la construcción de árboles de expansión mínima. Kruskal se centra en hacer selecciones óptimas a nivel local para construir el árbol, y su lógica evita la formación de ciclos, lo cual es crucial para este problema.
2. **Cola de Prioridad Ordenada por Coste**: Se empleó una cola de prioridad para manejar las aristas, que se ordena en base al peso de cada arista. Esta elección es fundamental para la eficiencia del algoritmo Kruskal, ya que permite acceder siempre a la arista de menor costo en cada iteración.
3. **Tabla Hash para Conjuntos de Nodos**: Se utilizó una tabla hash para mantener un registro de los conjuntos de nodos. Esto es en especial útil para verificar rápidamente que no se formen ciclos cuando se añaden nuevas aristas al árbol.
4. **Aristas totales = Numero de nodos** – 1: Una vez se obtiene una cantidad de (Numero de nodos – 1) vértices que no generen ciclos, el programa termina ya que, para todo árbol de expansión mínimo, este tendrá una cantidad de Cantidad de nodos – 1. Así evitando hacer verificaciones innecesarias posterior a eso.

En resumen, la combinación del algoritmo Kruskal, junto con estructuras de datos adecuadas como la cola de prioridad y la tabla hash, contribuye a la eficiencia de la solución propuesta. Estas elecciones permiten una implementación más rápida y eficiente, evitando trabajos innecesarios y acelerando el proceso de construcción del árbol.

Tabla de tiempos de ejecución archivos de prueba.

Matriz	Tiempo ejecución
3x3	9x10^-6
10x10	3.2x10^-5
100x100	0.000315
1000x1000	0.010951

Ejecución del código

Para poner en marcha el código, es imperativo operar en un entorno con el sistema operativo Linux y asegurarse de que todos los archivos .h y .cpp correspondientes a la solución desarrollada, así como los archivos .txt para pruebas, estén organizados en un mismo directorio. A continuación, se describe el proceso paso a paso:

1. **Preparación del Entorno:**
    - Abra una terminal de Linux y navegue hasta el directorio que contiene todos los archivos mencionados anteriormente.
2. **Compilación:**
    - Ejecute el comando make en la terminal para compilar todos los archivos necesarios para el programa.
3. **Ejecución del programa:**
    - Posteriormente, ejecute ./MST para iniciar la ejecución del programa.
    - Al comienzo, el programa solicitará, a través de la consola, el nombre del archivo que contiene la matriz de adyacencia.
    - Nota: en caso de que el archivo no exista, se entregará un mensaje de error por consola.
4. **Despliegue de Resultados:**
    - Si todo procede sin inconvenientes, el programa se ejecutará y presentará por consola, la siguiente información:
        - Cantidad de nodos que contiene el archivo ingresado.
        - Aristas que forman el Árbol de MST ya calculado.
        - Tiempo de ejecución al momento de llamar al cálculo de MST.
        - Cantidad de vértices totales que contiene el archivo.
        - Cantidad de vértices revisados hasta encontrar la solución final.

Librerías utilizadas:

El desarrollo del código implico el uso de diversas librerías, entre las que se incluyen <sstream>,<iostream>,<stdlib>,<ctime>,<vector>,<queue>,<unordered_map>, y <fstream>

Bibliografía

S., T. H. (2009). *Introduction to algorithms (Third edition.).*