Taller Nº 2

Clique Máximo de Mayor Tamaño

Taller de programación 1-2024

Fecha: 27/05/2024

Autor: Mohamed Al-Marzuk

TALLER N°2

Clique Máximo de Mayor Tamaño

Explicación breve del algoritmo

El problema consiste en hallar dentro de un grafo el clique máximo de mayor tamaño. Un clique máximo es un subconjunto de vértices de un grafo, que tienen la particularidad de estar conectados todos entre todos, y que además, no hay ningún otro vértice que se pueda agregar al conjunto de forma que siga siendo clique. Entonces, el objetivo es encontrar el mayor conjunto de vértices que cumplan esta propiedad. Para esto, se define una clase “Clique” para poder trabajar con el grafo, y se utilizará una lista de adyacencia para poder representar las aristas existentes dentro de él. Además, se define una variable de tipo entero que representa el tamaño del grafo, es decir, la cantidad de vértices, y otra variable que representa el tamaño del mayor clique máximo encontrado. Se utilizará el algoritmo de Bron-Kerbosch para poder realizar esta tarea, pero se le aplicarán ciertas modificaciones con el objetivo de optimizar el rendimiento del programa y reducir el tiempo de ejecución.

Heurísticas o técnicas utilizadas

Como se mencionó con anterioridad, el algoritmo a utilizar es el de Bron-Kerbosch, el cual utiliza una técnica de búsqueda recursiva para encontrar todas las combinaciones de vértices que forman cliques. Para realizar esto se utiliza Backtracking. Con el objetivo de optimizar este algoritmo, se utilizaron técnicas como la poda de los potenciales caminos que no pueden superar al clique máximo actual, el seguimiento del tamaño del clique máximo encontrado, actualizando ese número si es que se encuentra uno más grande, y se utiliza la intersección de conjuntos para considerar únicamente los vértices relevantes.

Funcionamiento del programa

A continuación se brindará un paso a paso del funcionamiento del programa:

1. El programa lee el archivo indicado por terminal y construye un grafo correspondiente al tamaño y las aristas indicadas en ese archivo. Luego, se crean los conjuntos P, R, X y C, y se rellena el conjunto P con todos los vértices del grafo.

2. Tras esto, se hace un llamado a la función bronKerbosch con los conjuntos P (vértices a explorar), R (vértices del clique actual), X (vértices ya explorados) y C (clique máximo).

3. Una vez en la función, la condición de salida es que tanto el conjunto P como el conjunto X estén vacíos, si es así, encontramos un clique, por lo que se analiza si este clique supera en tamaño al actual, si es así, se actualiza el tamaño y se borran todos los cliques que sean menores a ese tamaño, y si no, el clique simplemente se guarda.
4. Si no se cae en la condición de salida, lo que ocurre es que se va a elegir un vértice pivote del conjunto de P unido con X, siendo este el de mayor grado (el que tiene más vecinos), y se creará un nuevo conjunto de vértices quitando los vecinos del pivote.
5. Ahora se iterará sobre los vértices de este nuevo conjunto, y lo primero que se hace es determinar si merece la pena seguir explorando la rama (Si el tamaño de R + el tamaño de P son mayores que el tamaño del clique máximo actual), y si no, se poda.
6. Se crean los nuevos conjuntos para la siguiente llamada recursiva, R1 guarda el vértice actual, P1 intersecta a P con los vecinos del pivote y X1 intersecta a X con los vecinos del pivote.
7. Se llama a la función recursivamente, con P1, R1, X1 y C.
8. Cuando terminen todas las recursiones, se agregará v a X y se borrará de P. Se libera la memoria de P1, R1 y X1.
9. Se retorna C (el conjunto de todos los cliques máximos de mayor tamaño).
10. Se imprime por pantalla el primer clique del conjunto, junto a su tamaño.
11. Se libera la memoria de P, R, X y C.

Los resultados obtenidos se pueden ver en la siguiente tabla:

Nombre del archivo	Tamaño clique máximo	Tiempo de ejecución (s)
graph_6_0.800000_3.txt	22	0,0299
graph_7_0.800000_3.txt	23	0,0737
graph_10_0.100000_2.txt	20	0,0047
graph_10_0.100000_3.txt	23	0,0031
graph_10_0.300000_3.txt	24	0,0049
graph_10_0.750000_3.txt	28	0,0198
graph_32_0.947994_4.txt	26	0,3133
graph_33_0.979994_4.txt	35	0,0142

Aspectos de implementación y eficiencia

Para la implementación de este problema, se pueden destacar varios factores que ayudan a mejorar la eficiencia del algoritmo.

1. **Poda:** Se descartan todos los caminos que no puedan superar en tamaño al clique máximo encontrado, esto implica no recorrer caminos innecesarios que no puedan llevar a una solución al problema.
2. **Uso de punteros:** Se declaran todos los conjuntos como punteros, en vez de pasarlos como valor. De esta forma, se evita la creación de conjuntos cada vez más grandes con cada recursión, evitando el overhead de copia.
3. **Pivote:** Se escoge el vértice de mayor grado y se limita el espacio de búsqueda a los vecinos de este. Esto reduce el número de llamadas recursivas y el tamaño de los conjuntos en cada llamada.

La combinación de todo esto garantiza una implementación eficiente que permita reducir de forma considerable el tiempo de ejecución del algoritmo.

Ejecución del código

Para la realización de esta tarea se utilizó la Standard Template Library (STL), destacando las librerías vector y set, que sirvieron para operar con vectores y conjuntos.

Para poder ejecutar el código, es necesario utilizar el sistema operativo Linux, y asegurarse de que todos los archivos .h, .cpp y .txt estén en una misma carpeta, al igual que el archivo makefile, tras esto, se siguen los siguientes pasos:

1. **Preparación del entorno:** Abra una terminal Linux y navegue hasta el directorio de los archivos.
2. **Compilación:** Ejecute el comando make para compilar y crear todos los archivos necesarios para la ejecución del programa.
3. **Ejecución:** Invoque el comando ./main, y ahí se desplegará un menú. Pulse 1 para resolver un puzzle y escriba su nombre por terminal.
4. **Resultados:** Si el archivo existe, se mostrará el clique máximo encontrado, su tamaño y el tiempo de ejecución.