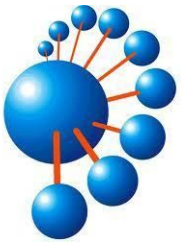

Universidad de Concepción

Departamento de Informática y Ciencias de la Computación

Grafos

Nombre: Soledad Vásquez.

Roberto Avila

Asignatura: Estructura de Datos

Profesor: Diego Seco

Ayudantes: Paulo Olivares

Alexander Irribarra

Fecha: 04 de julio de 2018

Índice

Introducción

Las conexiones por pares entre elementos desempeñan un papel fundamental en una amplia gama de aplicaciones computacionales. Las relaciones implícitas en estas conexiones conducen a una serie de preguntas naturales: ¿hay alguna manera de conectar un elemento a otro siguiendo las conexiones? ¿Cuántos otros artículos están conectados a un artículo determinado? ¿Cuál es la cadena de conexiones más corta entre este elemento y este otro?

La profesión del Ingeniero Civil Informático está evolucionando a un ritmo vertiginoso, abordando problemas cada vez más diversos y entretenidos. Una de las especialidades con más futuro es la del científico/ingeniero de datos, los cuales se ocupan de poner en valor el conocimiento que se puede extraer de los volúmenes de datos masivos que están disponibles hoy en dia. Existen muchos problemas que se pueden modelar empleando grafos, por tanto, conocer cómo modelar problemas mediante grafos utilizando algoritmos adecuados para resolver las preguntas de interés en un dominio , son herramientas fundamentales para un científico/ingeniero de datos.

Un Grafos es una estructura de datos , que contiene un conjunto de objetos llamados vértices unidos por enlaces llamados aristas , representando una relación binaria entre vértices.
Algunos modelos gráficos: grafos no dirigidos (con conexiones simples), dígrafos (donde la dirección de cada conexión es significativa)

Los grafos se utilizan para representar muchas aplicaciones de la vida real: se utilizan para representar redes. Las redes pueden incluir rutas en una ciudad o red telefónica o red de circuito, también son utilizados en redes sociales como linkedIn, Facebook, etc.
En este proyecto se modelará un grafo dirigido que es ingresado por el usuario , para luego analizar e inferir información interesante sobre el.

Descripción de las soluciones

En este proyecto se implementarán varios algoritmos para trabajar con los datos proporcionados , además se darán a conocer las estructuras de datos necesarias para ocupar exitosamente los algoritmos.

Se implementa un Tipo Abstracto de Datos (ADT), llamado Graph, proporciona las siguientes operaciones: Graph(int n)
int size() :Retorna la cantidad de vértices que contiene el grafo
int edges():Retorna la cantidad de aristas del grafo
void addEdge(int,int): Agrega dos vertices adyacentes , genera una arista entre ellos
pair<int,int> WCC() : Retorna la cantidad de vértices y aristas que contiene la mayor componente conexa
pair<int,int> SCC() :Retorna la cantidad de vértices y aristas que contiene la mayor componente fuertemente conexa
int transitive Closure():Retorna el promedio de vértices alcanzables por cada vértice

Esta implementación usará búsquedas en profundidad(DFS) y búsqueda por anchura(BFS), con Componentes fuertemente conectados, débilmente conectados y cierre transitivo.

La búsqueda en profundidad:

Es un método recursivo clásico para examinar sistemáticamente cada uno de los vértices y bordes en un gráfico. Para visitar un vértice:

1. Marcar como haber sido visitado.
2. Visita (recursivamente) todos los vértices que están adyacentes a él y que aún no han sido marcados.

Lista de adyacencia:

Se usa una matriz de listas vinculadas. El tamaño de la matriz es igual al número de vértices. Esta representación también se puede usar para representar un grafo ponderado. Los pesos de los bordes se pueden almacenar en nodos de listas vinculadas

Strongly Connected Component(SCC)

Un grafo dirigido está fuertemente conectado si hay un camino entre todos los pares de vértices.Un componente fuertemente conectado (SCC) de un gráfico dirigido es un subgrafo máximo fuertemente conectado.

Weakly Connected Component(WCC)

Dado un gráfico dirigido, un componente débilmente conectado (WCC) es un subgrafo del grafo original donde todos los vértices están conectados entre sí por algún camino, ignorando la dirección de los bordes.

Cierre Transitivo

La clausura transitiva es el grafo que resulta de añadir una aristas dirigida desde u hacia v si existe algún camino para ir desde u hacia v.

Vértice más conectado

La pregunta que se va a responder cuál es el vértice más conectado en el grafo osea el que tenga mayor suma entre los grados de salida y entrada. es útil esta información para saber cual es el punto donde se focaliza la mayor concentración de conexiones y así poner atención a un punto crítico en nuestro grafo.

Detalle de implementación

Aquí se expone lo necesario para implementar el código a fin de poder realizar una lectura ágil y a su vez para facilitar la lectura.

Archivo de referencia: GraphADT.h

Declaración de la clase ADTGrap.h que se realiza solo las especificaciones en: public que permite el acceso a tal término desde dentro y fuera de la clase y por otro lado el private que permite que sean accesibles por los propios miembros de la clase.

Archivo de referencia: Graph.cpp

En ese archivo se implementan los métodos declarado en Graph.h

El constructor Graph : Crea el grafo con capacidad de n vértices , el constructor emplea un arreglo de vectores que representa la lista de adyacencia , por tanto el índice del arreglo indica la lista de adyacencia del vértice i , además de es declarado un vector global de tamaño n , llamado visitado este guarda el estado de cada vértice (visitado /no visitado), además son almacenadas la cantidad de vértices y aristas a ingresar al grafo .

Destructores: Se encargan de eliminar todo el contenido del, esto se hace secuencialmente , haciendo delete a cada referencia contenida.

addEdge (int n , int v) : Agrega un vertice al grafo ,en donde n indica el inicio de la arista y v el final , siendo el sentido de la arista desde n a v , el método inserta en el arreglo de vectores en el índice n al final de su lista de adyacencia , y agrega una arista a la variable aristas.

recorrer grafo (int nodo , stack <int>* pila): es un dfs recursivo , mientras recorre el grafo va marcando los nodos visitado y cada nodo visitado lo apila en el stack.

dfs(int nodo , set<int>&s) :Este método mientras realiza un dfs recursivo , en un set guarda todos los nodos que recorre , para luego contar las aristas y los nodos , el criterio que es ocupado para contar aristas cada vez que se visita un nodo se agrega una y si se quiere recorrer un nodo que ya fue visitado se revisa en el set si este es parte del recorrido actual y si es así se agrega otra arista. Finalmente retorna un par que contiene los la cantidad de vértices visitado y la cantidad de aristas

reiniciarvisitados():este método hace que la lista de visitados pasen todos los nodos a no visitados.

transponerGrafo(): recibe de argumento de entrada un grafo , el grafo se transpone

cambiando el sentidos de las aristas.

Weakly Connected Component (WCC)

Componentes conexas en el grafo, retorna la cantidad de vértices y aristas de la componente conexa más grande. El método recorre el grafo sin importar el sentido de las aristas (grafo fundamental) , para así encontrar la componente conexa con más vértices y aristas.

se le entrega un set al método dfs , este se encarga de recorrer el grafo , contando las aristas y los vértices pertenecientes a la componente conexa. Compara cada cantidad de vértices retornada por dfs , retorna la con más vértices.

Strongly Connected Component(SCC)

Lo que hace este método es buscar la componente fuertemente con mayor cantidad de vértices y aristas , En este proyecto se ocupa el algoritmo de kosaraju el cual consiste en realizar un DFS al grafo e ir apilando en un stack (se ocupa el método recorrer grafo) los vértices visitados , luego se transpone el grafo por consiguiente recorre el grafo , empezando por el último vértice apilado , se hace pop de cada vértice y se hace un dfs si no está previamente visitado.

Al hacer el DFS se cuentan las aristas y los vértices de cada componente fuertemente conexa , conserva la con mayor cantidad de vértices y luego retorna un par con los vértices y aristas.

Cierre Transitivo

El metodo de cierre transitivo realiza un dfs por cada nodo ,y luego como el DFS retorna la cantidad de vértices recorridos , se hace una suma por cada DFS hecho y se divide por el total de nodos que contiene el grafo.

Vértice más conectado

En este método se recorre dos veces la lista de adyacencia para contar primero los grados de salida de todos los vértices , luego se cuentan todos los grados de entrada , finalmente se retorna el vértice con mayor sumatoria de grados .

Análisis Teórico

En el análisis teórico de peor caso podemos ver que al implementar con :

DFS O(n+m) con lista de adyacencia

- **Weakly Connected Component** O(n+m)
- **Strongly Connected Component** O(n+m)
- **Cierre Transitivo** O n*(n+m)
- **Vértice más conectado** O(n+m)

Análisis de espacio

Lista de adyacencia en el peor de los casos, puede haber número de aristas en un grafo que consume en espacio es O(n^2) , esto puede ocurrir cuando el grafo está muy conectado.

Conclusión

Para concluir, hay muchas situaciones en las cuales el modelado más conveniente de los datos de una aplicación es mediante grafos, por ejemplo la representación de una red de carreteras, calles, telecomunicaciones, electrificación, internet, planificación de tareas.
Los datasets proporcionados para dicho proyecto Amazon product co-purchasing network, los datos coinciden con lo que se implementó en este proyecto,tener la capacidad de analizar toda esa cantidad de información eficientemente, es favorable para deducir cosas sobre el grafo .
En este proyecto se expusieron varios problemas para trabajar con grafos ,tener la certeza de cuáles son las componentes fuertemente conexa más grande , el nodo más conectado , la componente conexa mayor , como el cierre transitivo, cambió la forma en como se puede ver un grafo .