Emilia Victoria Jácome Iñiguez.

A00828347

Estructura de Datos y Algoritmos Fundamentales

ITD

22/11/2020

Reflexión Actividad 4.3: Grafos

Realizar en forma individual una investigación y reflexión de la importancia y eficiencia del uso grafos en una situación problema de esta naturaleza, generando un documento llamado "ReflexAct4.3.pdf".

Un grafo es una estructura de datos en un representación abstracta de una serie de relaciones que contiene distintos nodos también denominados vértices los cuales están relacionados mediante aristas (o edges en inglés) los cuales pueden tener una dirección o no. Los gráficos son representados visualmente por una serie de puntos (sus nodos) conectados por líneas (aristas). La teoría de grafos tiene una importancia tanto para la rama de la Matemática Discreta como para el área de la ciencia computacional ya que es una manera efectiva de representar relaciones entre elementos y caminos para llegar de un determinado elemento a otro. (Universidad de Pamplona, 2012)

El primer acercamiento a la teoría de grafos fue desarrollada por Leonhard Euler en el año de 1736 en su intento por resolver el problema de los puentes de Königsberg. De igual manera, la teoría de grafos es considerada como uno de los primeros resultados topológicos de la geometría. Otras aplicaciones dentro del campo científico corresponden a la publicación de las leyes de circuitos eléctricos de Gustav Krichoff en 1845 y el problema de los 4 colores de Francis Guthrie en 1852 donde se plantea que con tan solo 4 colores es posible colorear cualquier mapa de países de modo que 2 países vecinos nunca podrán tener el mismo color. (Universidad de Pamplona, 2012)

Dentro de la programación es posible representar los grafos mediante listas y matrices de adyacencias. Se recomienda usar listas para grafos dispersos y matrices cuando se requiere un acceso rápido a las relaciones de cada nodo, aunque consumen grandes espacios de memoria, por lo que no se recomiendan cuando se tiene una gran cantidad de datos. (Universidad de Pamplona, 2012)

El uso de esta estructura en particular, se puede utilizar para desarrollar los siguientes algoritmos:

• Algoritmo de búsqueda en anchura (BFS)

• Algoritmo de búsqueda en profundidad (DFS)

• Algoritmo de búsqueda A*

• Algoritmo del vecino más cercano

• Ordenación topológica de un grafo

• Algoritmo de cálculo de los componentes fuertemente conexos de un grafo

• Algoritmo de Dijkstra

• Algoritmo de Bellman-Ford

• Algoritmo de Prim

• Algoritmo de Ford-Fulkerson

• Algoritmo de Kruskal

• Algoritmo de Floyd-Warshall

Fuente: (Universidad de Pamplona, 2012)

Aplicaciones en la vida real:

- Resolución de la síntesis de circuitos secuenciales, contadores o sistemas de apertura.
- Dibujo computacional.
- Modelación de trayectos de medios de transporte tomando como referencia las calles de una ciudada-
- Administración de proyectos específicamente para la técnica de PERTE donde se puede representar cada proyecto con un nodo y utilizar el método del camino más corto para optimizar el tiempo de concretación de cada proyecto.
- Se pueden modelar las conexiones de redes sociales donde se puede cuantificar y abstraer relaciones complejas para representar visualmente las relaciones sociales de un determinado perfil.
- En el estudio de la biología y el hábitat se estudian las relaciones entre cada especie con el de su hábitat, de modo que un grafo permite modelar a un ecosistema determinado.
- Aplicaciones para sistemas de información geográficos (Google Maps, Waze, Maps, Uber, UberEats, Glovo, etc).
- Enrutamiento de aviones, tráfico, trenes, autos.

- Trackeo de packetes, pedidos.

Fuente: (Universidad de Pamplona, 2012)

La importancia del uso e implementación de grafos dentro de la programación radica en ponerlos a prueba en la programación competitiva. Tal como su nombre lo indica, este concepto se refiere a la puesta en práctica del conocimiento de estructuras de datos, algoritmos y programación para resolver una serie de ejercicios que permiten desarrollar las habilidades de uno en la resolución de problemas complejos dentro de un concurso. Los algoritmos que se pueden crear con los grafos son una herramienta importante para la resolución de este tipo de problemas relativos a redes y cuadrículas. (Dubey, 2020)

La eficiencia en el uso de grafos se encuentra en que en lugar de estar recurriendo al uso de apuntadores que indiquen la relación de un nodo con otro en forma de cadena, la organización visual de las relaciones del grafo es mucho más simple de comprender y de recorrer el grafo acudiendo al uso de algoritmos más eficientes y de complejidad menor a $O(n^2)$ al utilizar una matriz o una lista de adyacencias construidas en base de arreglos dinámicos (vectores). (Apuntes de Grafos, 2006)

Tabla No.1: Análisis de Complejidad de Algoritmos Utilizados:

Nombre de la función:	***Descripción:***	***Complejidad:***
void copyVertices()	Método para copiar los vértices del vector vertexes al vector vertices	O(n)
Graph(stack<T>)	Constructor de la clase graph que crea la lista de adyacencias	O(n)
int findVertex(T vertex)	Devuelve la posición de un vertice dado dentro de la lista vertexes	O(n)
void print()	Despliega la lista de adyacencias	$O(n^2)$
void addAdj(T origin, T target)	Método para añadir una adyacencia a la lista de adyacencias	O(1)
void countAdj()	Método para contar las adyacencias de cada vertice	O(n)
void leerArchivo(Graph<Dato>&graph_IPS)	Leer el archivo fuente y construir el grafo de datos	O(n)

void heapSort(DoubleLinkedList<T> &dllist)	Ordenar los elementos del grafo en orden descendente según su cantidad de adyacencias	O(n)
void createKey(string IP)	Crear una llave de agrupación para cada IP	O(n)

Una ventaja del uso de grafos, es que existe una amplia cantidad de algoritmos que funcionan para todo tipo de arquitecturas de grafos, de modo que a pesar de que varios algoritmos sirven para efectuar el mismo método, unos pueden ser más eficientes que otros para un determinado tipo de situación lo que nos garantiza un margen de optimización en términos de complejidad, espacio de memoria utilizado y tiempo de ejecución. De igual manera, los grafos presentan un alto grado de flexibilidad en su estructura lo cual los hacen adaptables a cualquier tipo de dato, o arreglo de datos que se quiera utilizar (ya sea vectores, arreglos, linkedlists, stacks, queues, heaps, etc) lo que hace que su aplicación sea mucho más amigable con el usuario y más personalizable de acuerdo a la situación. Aunque por lógicas razones, se hizo uso de los vectores en esta actividad para garantizar un espacio dinámico en la aplicación donde el espacio utilizado se irá llenando dinámicamente conforme se vayan encontrando más incidencias y si no encuentra ninguna, el espacio se mantiene vacío sin utilizarse. (Herrera Jiménez, Salcedo Parra, & Gallego Torres, 2014)

Referencias:

Apuntes de Grafos. (2006). Recuperado el 22 de Noviembre de 2020, de Departamento de Informática de la Universidad de Valloid: https://www.infor.uva.es/~cvaca/asigs/estr0506apg.pdf

Dubey, S. (16 de Julio de 2020). *A Guide to Important Graph Algorithms for Competitive Programming*. Recuperado el 22 de Noviembre de 2020, de codeburst.io: https://codeburst.io/a-guide-to-important-graph-algorithms-for-competitive-programming-e59127634e2d

Herrera Jiménez, S. E., Salcedo Parra, O. J., & Gallego Torres, A. P. (31 de Enero de 2014). *Eficiencia algorítmica en aplicaciones de grafos orientadas a redes GMPLS*. Recuperado el 22 de Noviembre de 2020, de researchgate.net: https://www.researchgate.net/publication/304208451_Eficiencia_algoritmica_en_aplicaciones_de_grafos_orientadas_a_redes_GMPLS/fulltext/577d572608aeaa6988abadcf/Eficiencia-algoritmica-en-aplicaciones-de-grafos-orientadas-a-redes-GMPLS.pdf

Universidad de Pamplona. (11 de Julio de 2012). *Teoría de grafos*. Recuperado el 22 de Noviembre de 2020, de Universidad de Pamplona: http://www.unipamplona.edu.co/unipamplona/portalIG/home_23/recursos/general/11072012/grafo3.pdf