

Implementación de la clase Graph en C++*

*Junto a algunos algoritmos conocidos en la teoría de grafos no dirigidos.

Víctor Samuel Pérez Díaz
Matemáticas Aplicadas y Ciencias de la Computación
Universidad del Rosario
Bogotá D.C., Colombia
victor.perez@urosario.edu.co

Juan Esteban Murcia Nieto
Matemáticas Aplicadas y Ciencias de la Computación
Universidad del Rosario
Bogotá D.C., Colombia
juane.murcia@urosario.edu.co

Index Terms—grafo, algoritmo, nodo, arista, matriz

I. INTRODUCCIÓN

Este documento busca exponer una implementación de la clase Graph en el lenguaje de programación C++. Junto a esta, se realizaron varios algoritmos conocidos en la teoría de grafos no dirigidos. C++.

II. RESUMEN

Este proyecto se realizó con los ánimos de tener una representación de grafos no dirigidos. Junto a esta, también se realizaron varios algoritmos clásicos enseñados en nuestro curso de *Teoría de Grafos*, junto a algunos que investigamos por nuestra cuenta.

La clase Graph se basa en la matriz de adyacencia de un grafo no dirigido. Todos los algoritmos fueron construidos en base a esta.

Por tanto, la clase Graph cuenta con tan solo un parámetro *matrix*, en el cual se almacena la matriz de adyacencia.

Los métodos desarrollados fueron:

- Graph y sus derivados.
- uncolored_vertex
- colored_vertex
- colored_neighbor
- uv_path
- grade
- empty
- order
- size
- max_grade
- min_grade
- remove_node
- remove_edge
- add_node
- add_edge
- clear
- print_matrix
- find_path
- color_graph
- breadth_search

Identify applicable funding agency here. If none, delete this.

- depth_search
- shortest_uv_path

Cada uno de los anteriores será explicado en la sección **Funcionalidad**.

III. FUNCIONALIDAD

La clase Graph está basada en la matriz de adyacencia de un grafo no dirigido, esta siendo su único parámetro.

El primer acercamiento a la idea de la clase fue crear dos estructuras nodos (V) y aristas (E), la construcción resulta ser conflictiva al hacer uso de punteros, los cuales interactúan con la memoria del computador. Dado esto, encontramos que la implementación a través de la matriz resultaba mucho más práctica.

Se desarrollaron varias versiones del constructor Graph:

- Graph(): Constructor vacío, un grafo sin nodos ni aristas.
- Graph(Graph): Constructor copia.
- Graph(A_matrix): Construye un grafo con una matriz de adyacencia específica.
- Graph(filename): Construye un grafo con una matriz de adyacencia específica contenida en un archivo .txt.

También, se realizó el método destructor:

- ~Graph(): Destructor de la clase.

En la **sección privada** de la clase, se definió el parámetro:

- matrix: Matriz de adyacencia del grafo.

Y además, se definieron los siguientes métodos:

- uncolored_vertex: Función que determina si un vértice no tiene color dada una coloración.
- colored_neighbor: Función que determina si un vértice tiene un vecino con un color dado.
- uv_path: Función que determina si hay un *uv*-camino en el grafo.

Ahora, en la **sección pública**, se definieron los siguientes métodos (junto a los constructores y destructor de la clase, definidos arriba):

- grade: Función que calcula el grado de un vértice.
- empty: Función que determina si un grafo no tiene vértices.

- `order`: Función que nos da el número de vértices del grafo.
- `size`: Función que nos da el número de aristas del grafo.
- `max_grade`: Función que nos da el máximo grado del grafo.
- `min_grade`: Función que nos da el mínimo grado del grafo.
- `remove_node`: Función que remueve un vértice específico y todas sus aristas adyacentes.
- `remove_edge`: Función que remueve una arista específica del grafo.
- `add_node`: Función que inserta un vértice nuevo al grafo.
- `add_edge`: Función que inserta una nueva arista al grafo.
- `clear`: Limpia el grafo.
- `print_matrix`: Función que muestra la matriz de adyacencia del grafo.
- `find_path`: Función que determina si hay un camino euleriano en el grafo y en caso afirmativo, lo encuentra.
- `color_graph`: Función que da un coloreado apropiado a los vértices.
- `breadth_search`: Función que retorna un árbol de expansión del grafo usando el algoritmo de búsqueda a lo ancho.
- `depth_search`: Función que retorna un árbol de expansión del grafo usando el algoritmo de búsqueda a profundidad.
- `shortest_uv_path`: Función que usa el algoritmo de Dijkstra para encontrar el menor u, v -camino en el grafo.

IV. DESCRIPCIÓN

En esta sección, se describirán los algoritmos que consideramos más relevantes, además, dados los análisis hechos a algunos de ellos, indicaremos su complejidad si es el caso:

- `find_path`: Lo primero que hace el algoritmo es contar la cantidad de vértices que tengan grado impar, si estos son mayores a 2, retorna una lista vacía, de lo contrario toma como inicio uno de los vértices de grado impar (cualquier vértice si no hay vértices de grado impar) recorre una arista hacia otro vértice, elimina la arista que recorrió y repite el proceso hasta que llegue a un vértice sin más aristas, al llegar añadirá ese vértice al camino y se devolverá al anterior, repitiendo el proceso hasta que no se pueda devolver más.

Complejidad: $\mathcal{O}(N + E)$

- `color_graph`: Inicializa todos los nodos con color 0 (sin color) y mientras haya un vértice sin color, recorrerá todos los vértices, si el vértice actual no tiene color y ninguno de sus vecinos tiene ese color, le asignará ese color a ese vértice, si acaba la iteración y siguen habiendo vértices sin color, cambia el color y repite el proceso.

Complejidad: $\mathcal{O}(N^3)$

- `breadth_search`: Crea un nuevo grafo con la misma cantidad de vértices que el grafo de entrada, luego

recorrerá los vértices de ese grafo, revisará la conexión del vértice actual con el resto de vértices, y agregará una arista entre un par de vértices en el grafo de salida, si esa arista está en el grafo de entrada y no existe ya un camino entre esos vértices en el grafo de salida.

- `depth_search`: Crea un nuevo grafo con la misma cantidad de vértices que el grafo de entrada, luego recorrerá todos los vértices de ese grafo, si encuentra una arista en el grafo de entrada que no este en el grafo de salida y no haya un camino entre esos dos vértices en el grafo de salida, agregará la arista y avanzará al siguiente vértice.
- `shortest_uv_path`: Lo primero que hace es el algoritmo es determinar si ambos vértices pertenecen al grafo, si esto no ocurre, soltará un mensaje de error y retornará una lista vacía. De lo contrario inicializa una lista de valores con 0 en el vértice de inicio e infinito en el resto de vértices, un conjunto con los vértices que puedo chequear y una lista con el camino que se debe tomar inicializada con infinito. Mientras el vértice de llegada siga en el conjunto, selecciona el vértice con menor valor en la lista de valores, eliminará ese vértice del conjunto y recorrerá todos los vértices del grafo, si encuentra que uno de los nodos es adyacente con el nodo actual y el valor del vértice en la lista de valores más 1 es menor al valor actual del vértice de la iteración, actualizo el valor por el menor y repito. Una vez termine, armará el camino y lo retornará. **Complejidad:** $\mathcal{O}(N^2)$

V. CONCLUSIONES

ACKNOWLEDGMENT

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression “one of us (R. B. G.) thanks ...”. Instead, try “R. B. G. thanks...”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

REFERENCES

Please number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use “Ref. [3]” or “reference [3]” except at the beginning of a sentence: “Reference [3] was the first ...”

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors’ names; do not use “et al.”. Papers that have not been published, even if they have been submitted for publication, should be cited as “unpublished” [4]. Papers that have been accepted for publication should be cited as “in press” [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

REFERENCIAS

- [1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," *Phil. Trans. Roy. Soc. London*, vol. A247, pp. 529–551, April 1955.
- [2] J. Clerk Maxwell, *A Treatise on Electricity and Magnetism*, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
- [3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in *Magnetism*, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
- [4] K. Elissa, "Title of paper if known," unpublished.
- [5] R. Nicole, "Title of paper with only first word capitalized," *J. Name Stand. Abbrev.*, in press.
- [6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," *IEEE Transl. J. Magn. Japan*, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetism Japan, p. 301, 1982].
- [7] M. Young, *The Technical Writer's Handbook*. Mill Valley, CA: University Science, 1989.

IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove the template text from your paper may result in your paper not being published.