Universidad Católica Andrés Bello.

Algoritmos y programación III.

Profesor: Carlos Alonzo.

Natalia Velásquez. C.I: 27.703.520.

Carlos Doffiny S-V. C.I: 27.814.707.

Informe del proyecto sobre Grafos en Python.

1. Descripción general del programa.

El presente programa se encarga de, dado un grafo descrito en un archivo de texto, responder a la pregunta de si se trata de un grafo Hamiltoneano, Euleriano, ambos o ninguno. Además, en el caso de cumplir con alguna de las dos condiciones, describe el recorrido que se debe realizar en tal grafo para que contenga un ciclo Hamiltoneano, Euleriano, o ambos.

Para hacer esto, lee el archivo de texto llamado “grafos.txt” que tiene el siguiente formato, indicado por el docente en las especificaciones del proyecto: la primera línea contiene el número de vértices, y las líneas siguientes contienen las aristas que conforman este grafo, en la forma v1, v2. Es decir, dos números separados por una coma. Esto incluye vértices aislados, en el caso de colocar un único número.

Luego de la lectura del archivo, el algoritmo almacena en una lista anidada todos las aristas y sus vértices, así como el número de vértices que posee el grafo, crea también un diccionario que se encarga de almacenar el grado de cada uno de los vértices, y por último crea otro diccionario que contiene los vértices que son alcanzables por cada vértice, es decir, con los que posee una arista.

Lo primero que verá el usuario será el menú, en el cual puede elegir cuál de las tres opciones desea conocer: si el grafo ingresado es Hamiltoneano, Euleriano o ambas.

Una vez elegida la opción que el usuario desea conocer, el algoritmo prosigue a verificar si el grafo ingresado es conexo, condición que siempre verificará primero para determinar si puede continuar analizando el grafo, ya que, de no serlo, desestima entonces que pueda ser Hamiltoneano o Euleriano, debido a que es obligatorio que para que todo grafo posea un ciclo Hamiltoneano o Euleriano, este debe ser simple y conexo. Por otro lado, si verifica que se trata de un grafo conexo, empieza a evaluar las condiciones que determinan su estatus como grafo Euleriano o grafo Hamiltoneano, dependiendo de la opción introducida por el usuario.

Por último, en pantalla se muestran los resultados, imprimiendo además el recorrido de sus ciclos respectivos en caso de ser verdadera la condición.

2. Descripción de las estructuras generales.

En el programa se hace uso de diversas estructuras que facilitan el manejo de la información contenida en los grafos ingresados por medio del archivo de texto, que nos permiten modelar los mismos y trabajar con ellos de la forma más sencilla y rápida posible.

Algunas de ellas, las más resaltantes, son:

a) **grafo:** Es la lista general donde se almacena la información general del grafo obtenida en la lectura del archivo. Es la lista principal y con ella se trabajan todas las funciones, es decir, que podría decirse que es la estructura más importante del programa ya que en ella es como modelamos cada grafo. Está compuesta por listas anidadas en cada posición. La posición 0 de grafo contiene el número de vértices que posee el grafo, y de ahí en adelante cada posición almacena una arista, si la posición posee dos vértices, o un vértice aislado si solo está almacenado un único vértice.

b) **gradosV:** Es un diccionario que contiene como claves cada uno de los vértices, y cuyos valores son el grado de cada uno de ellos. Es una estructura básica e importante para la verificación de las condiciones que hacen a un grafo Hamiltoneano o Euleriano.

c) **verticesRecorridos:** Si existe un ciclo Hamiltoneano, esta lista se encarga de almacenar los vértices en el orden en que conforman el ciclo para poder mostrarlos en pantalla. A pesar de ser una lista, cabe destacar que el algoritmo la trabaja como una pila, ya que solo apila y desapila por un extremo de la lista.

d) **aristasRecorridas:** De existir un ciclo Euleriano, esta lista se encarga de almacenar todas las aristas del recorrido, en el orden que satisface el ciclo Euleriano para mostrarlo en pantalla.

e) **verticesVisitables:** Es un diccionario que contiene como claves a cada uno de los vértices, y cuyos valores son listas que contienen todos los vértices adyacentes de un vértice en específico, es decir todos los vértices que están unidos a él con una arista.

3. Descripción de las funciones.

a) **Función "leerArchivo":** Esta función, como indica su nombre, se encarga de leer toda la información que contiene el archivo .txt. Recibe como parámetros al archivo que se va a leer y que ya fue abierto anteriormente, y a la lista grafo. Es importante resaltar que dicho archivo.txt debe estar guardado en la misma carpeta en donde se guarda el programa, y ese archivo debe llevar el nombre de grafos.txt, o en su defecto si se desea usar un archivo con un nombre diferente, el mismo debe estar igualmente guardado en la misma carpeta del programa, y el nombre puede ser modificado en la línea 273 del código.

Lo primero que hace esta función es leer el archivo completo línea por línea, almacenando cada una de ellas en diferentes posiciones de una lista llamada lectura. Pero esta lista contiene todos los datos del archivo en string, por lo cual va revisando posición por posición y carácter por carácter para identificar los números de cada vértice y de cada arista, para convertirlos en integer. Cabe destacar que durante este proceso el algoritmo descarta aquellos vértices que sean negativos, menores o mayores que el número de vértices del grafo. Verifica si una línea o posición de la lista lectura es un vértice aislado o una arista, y comprueba que no haya aristas o vértices aislados repetidos.

A medida que los números van pasando a integer, se van almacenando en sublistas que luego son anidadas a la lista grafo que es la principal y como modelamos cada uno de los grafos, de manera tal que la primera posición de la misma sea el número de vértices que posee el grafo, y el resto de posiciones representen a las aristas entre los vértices del grafo, o en su defecto, los vértices aislados.

Para finalizar la función retorna la lista grafo, que será la estructura más utilizada por el programa.

b) **Función "gradosVertices":** Esta función se encarga de indicar el número de aristas que posee cada uno de los vértices, es decir, su grado. Recibe como parámetros la lista grafo y el diccionario gradosV.

Lo primero que hace la función es crear en el diccionario gradosV tantas claves como vértices tenga el grafo, para luego almacenar como valor de cada una el grado de los mismos. Esto lo realiza

recorriendo toda lista grafo, y si identifica una arista, le suma 1 al grado de cada uno de los vértices que inciden sobre dicha arista. Al igual que si se trata de un vértice aislado no suma nada ya que por definición un vértice aislado posee grado 0.

Por último, la función retorna el diccionario gradosV que ahora posee la información que el algoritmo necesitará utilizar inteligentemente más adelante.

c) **Función "visitarBP":** Esta función se trata del algoritmo de Búsqueda en Profundidad visto, analizado y estudiando en clase. Recibe como parámetros el diccionario vértices (el cual tiene como clave todos los vértices del grafo y como valores 0 si el vértice es de color blanco, o 1 si es de color gris), una lista llamada pila (que será la usaremos como pila, apilando y desapilando los vértices solo por un extremo), la lista grafo, y una variable de tipo integer llamada numVertices que como dice su nombre, indica el número de vértices que posee el grafo.

Básicamente la función consta de la segunda parte del algoritmo de Búsqueda en Profundidad, que es un while, que dice que mientras la pila no esté vacía, irá recorriendo los vértices adyacentes del vértice que este en el tope de la misma, y si al de sus vértices adyacentes es de color blanco, apilará dicho vértice blanco. Y si no encuentra ninguno, entonces desapilará el tope.

Cabe destacar que esta es la única función de todo el programa sin retorno.

d) **Función "esGrafoConexo":** Esta función permite identificar si el grafo introducido es conexo o no. Recibe como único parámetro la lista grafo.

Para identificar si el grafo es conexo o no, esta función hace uso del algoritmo de Búsqueda en Profundidad, por lo cual primero crea la pila y crea el diccionario vértices, que tendrá como claves todos los vértices del grafo, y que sus valores para iniciar serán 0, que significa que el vértice es de color blanco, y si cambia a 1, es porque ahora es de color gris. Dentro de esta función se encuentra la primera parte del algoritmo de Búsqueda en Profundidad, que es un for que recorre el conjunto de vértices, y que, si encuentra uno en blanco, es decir con valor 0, llama a la función "visitarBP".

Por último la función posee un contador llamado numConexas que se inicializa en 0 y que se le suma 1 cada vez que se realiza un llamado a la función "visitarBP", ya que esta variable representa el número de componentes conexas del grafo. Y En caso de poseer una sola componente conexa,

determina que el grafo es conexo y retorna True. De lo contrario, si encuentra más de una componente conexa, entonces el grafo es disconexo y devuelve False.

e) **Función "sonVerticesVisitables":** Esta función crea un diccionario en donde las claves son todos los vértices del grafo, y sus valore son una lista con todos los vértices que son alcanzables desde él, es decir, que poseen una arista en común. Recibe como único parámetro la lista grafo.

Como se explicó en el párrafo anterior, básicamente la función crea al diccionario verticesVisitables, que luego lo retorna una vez que es creado y llenado con todos los datos pertinentes.

f) **Función "esGrafoEuleriano":** Esta función verifica si el grafo es Euleriano o no. Recibe como parámetros el diccionario gradosV, la lista grafo y la lista aristasRecorridas, que es la que tendrá el ciclo Euleriano en caso de poseer alguno.

La función tiliza como base el Teorema de Euler, que establece que si todos los vértices de un grafo tienen grado par, entonces podemos asegurar la existencia de un ciclo euleriano, y por ende, es un grafo Euleriano. Si no cumple con este teorema, automáticamente queda descartada la existencia de dicho ciclo y por ello retorna False al no ser Euleriano.

De cumplir con el Teorema, la función genera el recorrido de las aristas que se realiza en el grafo para obtener un ciclo Euleriano que incluya a todas las aristas, es decir, un grafo Euleriano, y por ende retorna True.

g) **Función "esGrafoHamiltoneano":** Esta función permite verificar si un grafo es Hamiltoneano o no. Recibe como parámetros el diccionario gradosV, la lista grafos y la lista vértices recorridos, que es la que va a contener los vértices en el orden en que son recorridos por el ciclo Hamiltoneano, en caso de que exista uno.

Lo primero que hace la función es verificar que todos los vértices tengan grado mayor o igual a 2, porque de no cumplir con esta condición, automáticamente se puede descartar que el grafo sea Hamiltoneano debido a que todos los vértices que forman parte del ciclo deben tener mínimo dos aristas que inciden entre sobre él, ya que dicho ciclo precisamente pasará por dos y solo dos de sus aristas.

Para identificar la existencia de un ciclo Hamiltoneano o no en el grafo, el algoritmo va a recorrer todos los posibles caminos hasta encontrar uno o no, pero antes de empezar a hacerlo, tiene que identificar cuál es el vértice que tiene menor grado en todo el grafo, para iniciar la búsqueda del ciclo desde ahí. Y ¿Por qué se busca el del menor grado?, pues se busca el de grado menor debido a que es el que tiene menos caminos posibles, lo que aumenta el porcentaje de probabilidad de acierto de cada camino, ya que, por ejemplo, si un vértice tiene grado 2, entonces es 100% seguro que, si existe un ciclo Hamiltoneano, el mismo pasará obligatoriamente por las dos aristas que inciden en ese vértice. En cambio, si un vértice es de grado 4, cada camino tendrá apenas un 50% de probabilidades de que forme parte de las 2 aristas de ese vértice que formarían parte del ciclo, en caso de que exista.

Una vez identificado el vértice con grado menor, es decir, el inicio del recorrido y del posible ciclo, el algoritmo entra en un While True que solo se acabará cuando se llegue a la conclusión de si el grafo posee o no el ciclo Hamiltoneano.

Dentro del while, lo que irá haciendo es identificar cuál es el vértice sobre el cual estamos en el recorrido, y ver cuáles son sus vértices visitables, los cuales se estudian y analizan para determinar cuál es una opción viable y cual no, ya que no se puede ir a un vértice ya recorrido, ni tampoco si existe una arista prohibida entre ellos (concepto que ya explicaremos más adelante). Luego de guardar dichas mejores opciones dentro de la lista mejorOpcion, se busca cuántas opciones hay disponibles, ya que, dependiendo de la cantidad de ellas, se tomarán las siguientes decisiones:

- 1 sola opción disponible: En este caso como solo existe un vértice al cual podemos ir, simplemente iremos a él y ya, y el algoritmo vuelve a darle una vuelta más al While para repetir todo de nuevo.
- 2 o más opciones: Si existen múltiples opciones, nos volveremos a apegar al concepto de buscar cuál de esos vértices tiene el menor grado, ya que será el que tenga los caminos con mayor posibilidad de acierto de estar en el ciclo, si este existe, y que por ende hará la búsqueda de los caminos de una forma mucho más eficiente. Pero puede ocurrir el caso en que el menor grado sea igual para dos o más vértices dentro de la lista mejorOpcion, por lo cual esto lo vamos a llamar “División”, ya que ambos caminos, al tener el mismo número de caminos o de aristas, tienen el mismo porcentaje de acierto.

Es sumamente importante reconocer e identificar este aspecto, ya que una división puede ser lo que haga que un grafo no posea un ciclo Hamiltoneano. La primera división que encontremos será la más importante de todas, y es la que llamaremos división principal, y el vértice de división será el vértice en el cual encontraron dos o más opciones con igual menor

grado. En este caso el algoritmo identifica el número de caminos que posee la división principal, y va a recorrer el primero, siguiendo todo lo explicado anteriormente vuelta a vuelta en el while. Si el recorrido por uno de los caminos no concluye con éxito al no encontrar un ciclo, el algoritmo irá regresando, hasta volver al vértice de la división principal, y si aún no hemos recorrido uno de los caminos disponibles de la división principal, continuaremos el recorrido por dicho camino. Pero si al final volvemos al vértice de la división principal y ya recorrimos todos sus posibles caminos, entonces el algoritmo retorna False ya que ha recorrido todos los posibles caminos del grafo y ninguno es un ciclo Hamiltoneano, básicamente ya no hay a dónde ir, lo que quiere decir que no hay forma ni manera que ese grafo tengo el ciclo que el algoritmo está buscando.

En el siguiente punto se va a explicar bien el concepto de aristas prohibidas, pero es importante mencionar también que, si se recorre un camino de la división principal, y dicho camino fracasa hasta volver al vértice de la división principal, antes de empezar a recorrer el siguiente camino, en caso de que exista, se borra todo el contenido de la lista aristasProhibidas, ya que, que una arista sea prohibida, no significa que no pueda formar parte del recorrido de otro camino diferente, ya que por ejemplo se puede llegar a la UCAB desde diferente lugares con tramos de caminos similares, como por ejemplo, la autopista, pero como se parte de distintos lugares, habrá tramos únicos a cada recorrido.

Por último, existen otras dos opciones de división. Ya se mencionó que la más importante es la primera que se encuentra, la cual se llama división principal, pero si se encuentra una después de esa, no importa tanto ya que, si los recorridos fracasan, las aristas se van bloqueando hasta volver a la división principal o hasta encontrar un ciclo. Además, existen casos en los que nunca se encuentra una división en todo el grafo mientras se analizan las diferentes opciones de cada vértice, y es por ello que el vértice inicial del recorrido, que es el que tiene el menor grado de todo el grafo, será inicializado antes del while como el vértice de división principal, y permanecerá así hasta que se encuentre la primera división, y en caso de que no, hasta que se concluya si el grafo posee o no un ciclo Hamiltoenano. Esto quiere decir que el vértice inicial adquirirá las mismas funciones y características del vértice de división principal, mientras no se sepa de la existencia de otro.

- Ninguna opción: si luego de analizar los vértices visitables del vértice sobre el cual estamos en el algoritmo, no posee ninguna mejor opción ya que o ya están en el recorrido o que las aristas que van hacia ellas están prohibidas, hay que analizar si esto quiere decir que encontramos una cadena, un ciclo o que simplemente llegamos a un callejón sin salida. Para ello, el algoritmo primero verifica la longitud de la lista

verticesRecorridos, y si es igual al número de vértices del grafo, quiere decir que estamos ante una cadena o un ciclo Hamiltoneano, y para verificar si es este último, el algoritmo analiza si el último vértice del recorrido posee una arista que lo conecte con el primer vértice que fue donde todo empezó, y si es así habremos hallado la cadena cerrada (ciclo) que determina que el grafo si es Hamiltoneano, por lo cual la función retorna True.

Si esta arista no existe, entonces habremos encontrado una cadena Hamiltoneana, pero no nos sirve para concluir nada, así que al igual que cuando el recorrido se encuentre en un callejón sin salida, empezará a retroceder vértice por vértice para volver a hacer todo el análisis y estudio que se lleva a cabo dentro del while. Y en caso de que se retroceda un vértice, la arista que unía al vértice que acabamos de retroceder hasta el vértice que acabamos de abandonar en el recorrido, será almacenado dentro de la lista aristaProhibidas, ya que ya el algoritmo sabe que esa arista ya se recorrió y que por ahí no se llega a un posible ciclo Hamiltoneano. Cabe destacar que una arista se bloquea en un solo sentido, es decir, que si la arista [4,5] está prohibida, el recorrido del posible ciclo no puede ir al vértice 5 desde el vértice 4, pero si podría ir desde el vértice 5 hasta el vértice 4, ya que para bloquear ese camino, la arista prohibida debería ser [5,4]