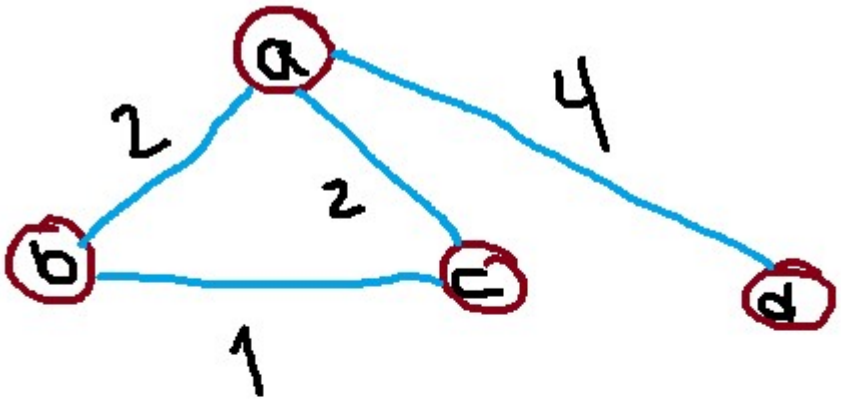
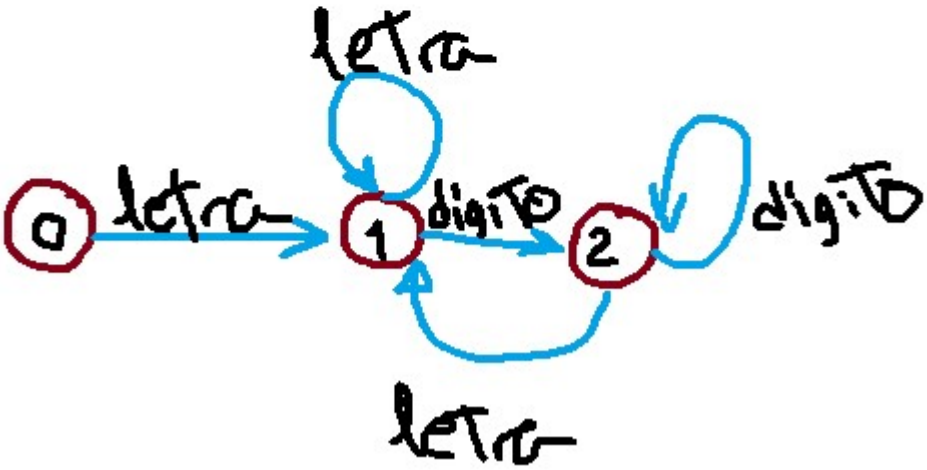
TDA Grafo

En ciencias de la computación, un grafo es un tipo de dato abstracto que está destinado a implementar grafos dirigidos y no dirigidos, y los conceptos de la teoría de grafos.

Un grafo consta de un conjunto finito de vértices (también llamados nodos), junto con un conjunto de pares desordenados de estos vértices para un grafo no dirigido, o un conjunto de pares ordenados para un grafo dirigido. Estos pares se conocen como aristas, y para un grafo dirigido también se conocen como flechas.

En un grafo se pueden asociar a cada arista algún valor, como una etiqueta o un atributo numérico (costo, capacidad, tiempo, etc.).





En la figura anterior, muestro dos ejemplos de grafos, o gráficas. La primera es una grafo no dirigido, y tiene los siguientes vértices V={a,b,c,d}, y sus aristtas son {<b,a>,<b,c>,<c,a>,<a,d>}, y es un subconjunto de VxV.

La segunda gráfica representa un autómata finito determinístico, que permite reconocer nombres válidos de identificadores en lenguajes como Pascal, o C. Por ejemplo, si estoy en el estado 0, y recibo de entrada la cadena “max12val2”, eventualmente llegaré a cualquiera de los estados 1 o 2 y habré terminado de leer la cadena. Si eso ocurre, la cadena es válida. Esto es, del estado 0 paso al estado 1, porque la entrada es la letra ‘m’, luego regreso al esto 1, porque la siguiente entrada es la letra ‘a’, si seguimos este procedimiento llegaremos al estado 1 y la entrada será el segundo 2, lo que nos llevará al estado 2, habremos terminado de recorrer toda la cadena, por lo tanto aceptaremos como válida la cadena. Si queremos analizar la cadena “2max3”, no podremos llegar a ningún estado de terminación (1,2), porque como iniciamos en el estado 0 y la entrada el el ‘2’, no podremos pasar al estado 1, porque para eso necesitamos una letra. Esa cadena no puede ser aceptada por nuestro autómata. Lo importante, para nosotros, del ejemplo de la segunda gráfica, no es que sea un autómata finito determinístico, sino que este es posible ser representado mediante una gráfica dirigida.

A continuación, muestro la especificación del TDA Grafo:

Para poder definir la signatura de las funciones u operaciones del TDA Grafo, debemos de considerar los siguientes tipos de datos:

ElemVertice: este tipo de dato define los valores con que podemos etiquetar o nombrar a los

vértices.

ElemArista: este tipo de dato define los valores que podemos asignar a las aristas.

Los tipos Grafo, Vertice, Arista, se explican por si mismos.

Espec Grafo

nuevoG: -> Grafo;

agregaVertice: Grafo, Vertice -> Grafo;

contieneAlVertice: Grafo, Vertice -> Booleano;

adyacente: Grafo, Vertice, Vertice-> Booleano;

vecinos: Grafo, Vertice -> Lista;

eliminaVertice: Grafo, Vertice -> Grafo;

agregaArista: Grafo, Vertice, Vertice -> Grafo;

eliminaArista: Grafo, Vertice,Vertice -> Grafo;

obtenValorVertice: Grafo, Vertice -> ElemVertice;

inicializaValorVertice: Grafo, Vertice, ElemVertice -> Grafo;

obtenValorArista: Grafo, Vertice, Vertice -> ElemArista;

inicializaValorArista: Grafo, Vertice, Vertice, ElemArista -> Grafo;

Axiomas: Grafo g; Vertice x, y; ElemVertice z; ElemArista w.

Tarea: define los axiomas.

Implementación:

Para poder definir el tipo Grafo, primero debemos de definir como representar el grafo. Una forma muy común es mediante una matriz de adyacencias. Para la primera gráfica del ejemplo, su matriz de adyacencias seria la siguiente:

	a	b	c	d
a	0	1	1	1
b	1	0	1	0
c	1	1	0	0
d	1	0	0	0

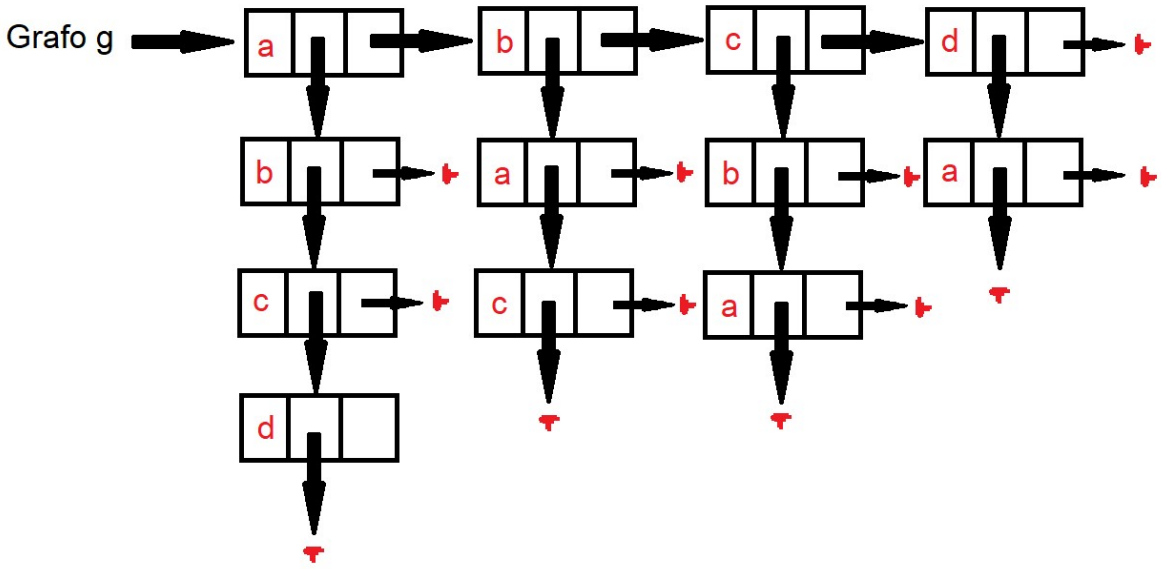
Como el grafo no es dirigido, tengo un 1 para la arista que va de ‘a’ a ‘b’ y otro 1 para esa misma arista pero de ‘b’ a ‘a’, y así para las demás aristas.

La segunda gráfica, o grafo, si es dirigido, por lo tanto no se repiten los 1’s, como en la primera gráfica. Su matriz de adyacencias es:

	0	1	2
0		Let	
1		Let	Dig
2		Let	Dig

Nota que en ambas matrices, el origen son los vértices de primera columna, y el destino los vértices del primer renglón.

Otra forma de representar un grafo, o gráfica, es mediante una lista de listas. La lista de adyacencias del primer grafo sería:



En esta representación vemos que la lista formada por los nodos horizontales corresponde a los vértices del grafo. Cada uno de esos nodos está asociado a otra lista, cuyos nodos contienen los vértices a los que están conectados mediante una arista. Desde el punto de vista de implementación en C, el tipo grafo no es otra cosa que una lista generalizada, esto es:

typedef ListaGen Grafo.

La tarea consiste en implementar todas las operaciones definidas en la especificación del TDA Grafo. Luego, implementar una función que reciba como argumento un Grafo g, y devuelva una lista l con los nodos que forman un camino hamiltoniano dentro de g. Complementariamente, hacer otra función que también reciba un Grafo g, y devuelva una Lista l con las aristas que forman un camino euleriano. Si se te complica definir estas dos funciones, búscalas en internet e impleméntalas usando las funciones de grafos que tú implementaste como primera actividad de la tarea. Implementar estas funciones quizá parezca complicado, pero no lo es. Con las actividades de esta tarea, doy por terminado el curso. No obstante, que las últimas cuatro estructuras no te las pude explicar de forma presencial, tu debiste de ser capaz de entenderlas y haber realizado las prácticas que deje. Durante el periodo presencial, expliqué con detalle las herramientas algebraica y de programación, necesarias para comprender e implementar todas las estructuras de datos que estudiamos, de forma presencial y no-presencial. Como de costumbre, deberás de enviar el código correspondiente, y la explicación para probarlo, al correo que hemos ocupado para tal efecto.

A continuación, transcribo las definiciones de caminos hamiltoniano y euleriano.

Un camino hamiltoniano, en el campo matemático de la teoría de grafos, es un camino de un grafo, una sucesión de aristas adyacentes, que visita todos los vértices del grafo una sola vez. Si además el último vértice visitado es adyacente al primero, el camino es un ciclo hamiltoniano.

En la teoría de grafos, un camino euleriano es un camino que pasa por cada arista una y solo una vez. Un ciclo o circuito euleriano es un camino cerrado que recorre cada arista exactamente una vez.