UNIVERSIDAD NACIONAL
COSTA RICA

Facultad de Ciencias Exactas y Naturales

Asignatura:

Paradigmas de Programación

PROYECTO 2: BIBLIOTECA DE GRAFOS PROLOG

Profesor:

Eddy Miguel Ramírez

Estudiantes:

Jonathan Estrada Vargas
José Isaac Zeledón Jiménez
Andrés Jiménez Elizondo

II CICLO

2019

Índice

1. Introducción

En el siguiente se describe la solución que se le dio a el problema planteado por el profesor, el cual fue el de realizar una biblioteca de grafos en el lenguaje de programación PROLOG, el que se maneja en el paradigma de programación lógico, esto para conocer el otro paradigma que el curso evalúa.

Además de que se puede notar la diferencia de Prolog con los lenguajes de paradigma funcional que también se evaluaron en este curso, ya que como se puede visiblizar PROLOG en sintáxis se parece mucho a el lenguaje de programación funcional concurrente Erlang, pero en función de los métodos que se pueden desarrollar y su funcionamiento con una pila es lo que lo diferencia y destaca como el eligido para resolver en muy pocas líneas de código problemas de backtracking. Con la anterior afirmación podemos notar el porque el profesor nos pone esta asignación ya que muchos de los algortimos a implementar en esta bilbioteca de grafos hacen uso de esta técnica, y con la ventaja de que PROLOG su compilador funciona como una pila, entonces el detalle de la solución de estos algoritmos es colocar los casos de parada y las condiciones correctas para solucionar los métodos planteados en la descripción del proyecto.

Añadido a esto anterior nos da la oportunidad de reforzar conceptos, "vistos" en asignaturas anteriores, ya que se debe de entender una basta gama de conceptos de teoría de grafos, para poder resolver los ejercicios en PROLOG. Pero este repaso de conceptos y algoritmos nos ayuda a que no sólo se pueda resolver los problemas en PROLOG, sino que al repasar estos conceptos desde la teoría de grafos, se puede asegurar que a nivel algorítmico esto queda claro, por consiguiente, se podría resolver en cualquier lenguaje de programación, con la salvedad de que se podría presentar limitación con características particulares del lenguaje como ejemplo se puede poner: el manejo de punteros en C++.

Es por esto que se puede notar que para la especificación de este proyecto se cambió el formato de en el que se presenta la documentación del mismo ya que al incluir un Marco Teórico se da la oportunidad de en este desarrollar con más profundidad todo lo relacionado a la teoría de grafos, así que se puede adelantar que en esta sección de la documentación se aprovecho para el desarrollo de la teoría concerniente a los métodos solicitados, además de que se hará referencia a la clase que el profesor decidió desarrollarla sobre temas básicos de grafos y árboles para así reforzar la materia "previamente conocida".

Expresando el sentir de los miembros del grupo se puede mencionar que este proyecto fue bastante difícil de desarrollar ya que el lenguaje de programación PROLOG es una herramienta complicada de trabajar, más esta versión, gprolog mantenida por Daniel Diaz. Pero esto probó la capacidad del equipo para adaptarse a la herramienta e intentar solucionar los problemas presentados durante el desarrollar la solución. Además nos tomamos la libertad de también expresar el sentir general de acerca del curso debido a que este es el último reporte escrito que se entregará, podemos decir que este curso aunque complicado fue gratificante ya que nos enseñó la importacia de saber de gramáticas regulares, lenguajes regulares, gramaticas libres de contexto y lenguajes libres de contexto, ya que no solo es materia sin utilidad sino que es una parte nuclear en la que es la computación y los lenguajes de programación los cuales son las herramientas con las que nos vamos a desarrollar en el campo profesional, pero con los paradigmas fucional y lógico que se vieron recordamos que las herramientas y lenguajes en cualquier momento desaparecen o se quedan obsoletas, y lo que importa es el expresar en un lenguaje de programación, el que sea, las funciones y relaciones que expresan el conjunto de pasos de un algortimo.

2. Marco Teórico

2.1. ¿Qué es un Grafo?

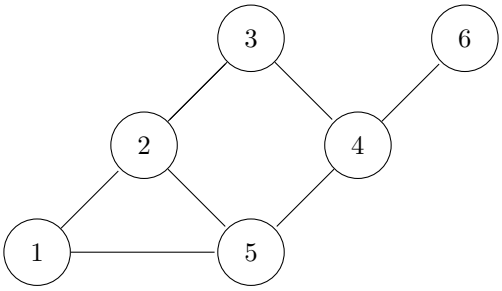
Antes de iniciar se deberia de introducir el concepto de estructura de datos el cual es algo que el profesor recurrentemente recuerda en clase a la hora de que se habla de grafos, arboles, pilas o cualquier otra estructura de datos.
Así que de esta manera citando a el profesor Eddy Ramírez una estructura de datos es:

> "mecanismo para adminstrar datos, y donde administrar quiere decir, insertar, buscar eliminar datos y modificar datos...La forma en la que inserto y modifico va ha ser distinto en cada estructura de datos ya que las estructuras son distintas en si mismas. La estructura de datos existe independientemente de que yo la implemente o no." [1]

Con esto se evidencia el objetivo de reforzar conceptos de los cursos de estructuras discretas y estructuras de datos, ya que con esta introducción se procedió a hacer la introducción a el concepto de grafo el cual fue introducido por el profesor como:

> "una tupla de nodos y aristas, donde las aristas son relaciones binarias entre los nodos, la condición para que sea un grafo simple es que no haya una relación reflexiva, osea que no haya una relación de un nodo con sí mismo. El grafo es la estructura de datos más simple." [1]

[7] Para complementar la definición que el profesor brindó podemos añadir que un grafo es un conjunto de objetos llamados vertices o nodos unidos por enlaces llamados aristas o arcos. Son objeto de estudio en la teoría de grafos. Tipicamente se reprensenta de la siguiente forma:



2.1.1. Camino

Este es uno de los conceptos básicos de la teoría de grafos y según el profesor Eddy un camino se define como:

> "un camino es una secuencia de nodos."

El concepto de secuencia es importante em la definición de camino ya que me da la premisa de que esta posee un orden, además de que posee una característica más, lo que es que para todo i (n_i, n_{i+1}) es elemento de las aristas. Esto quiere decir que el par de dos elementos consecutivos cualesquiera es elemento de la relación.

Para reforzar el concepto brindado por el profesor agregaremos la definición de camino que se nos brinda en otras fuentes que basicamente es la misma, ya que en otras fuentes se dice que un camino es un conjunto de vértices interconectados por aristas, dos vértices están conectados si existe un camino entre ellos.

2.1.2. Grado

Este concepto esta relacionado con los nodos de un grafo, ya que este concepto esta dado por el máximo grado de sus nodos, así mismo el grado de un nodo es la cantidad de aristas que ese nodo tiene.
En un grafo dirigido existen 2 tipos de grado:

- Grado de incidencia.

- Grado de salida.

Grado de incidencia refiriéndose a las aristas que llegan a ese nodo.
Grado de salida refiriéndose a las aristas que salen de ese nodo.

Como ejemplo práctico el profesor explicó que una red se puede ver como un grafo, y que originalmente Google, tenía modeladas la páginas como un grafo dirigido, ya que una página incidía en otra pero que no necesariamente está ultima tuviera una arista a la anterior, y así con esto, originalmente se podía manejar el page rank de Google con el grado de incidencia de las páginas, ya que se colocaba como relevante si la página tiene mucha incidencia.

El grado de un grafo es importante también para determinar si en un grafo existe o no camino de euleriano, ya que un grafo es euleriano si todos los nodos poseen grado par, ya que con esta condición puedo empezar cualquier nodo para encontrar un camino euleriano. Y en un grafo semieuleriano solo voy a encontrar dos caminos eulerianos, todo esto también se relaciona con el grado ya que este grafo está descrito como que todos sus nodos poseen grado par salvo 2 que poseen grado impar, y para encontrar los caminos eulerianos se debe de iniciar en estos nodos de grado impar.

2.1.3. Ciclo

Un ciclo es una sucesión de aristas adyacentes, donde no se recorre dos veces la misma arista, y donde se regresa al punto inicial.
[9]

2.2. Conexo

Un grafo es un grafo conexo en el que para cualesquiera 2 nodos A y B en el grafo existe un camino de A a B, esto para grafos no dirigidos. Es posible detgerminar si un grafo es conexo con los algoritmos de busqueda a la anchura (BFS) por sus siglas en inglés o con el algoritmo de busqueda a profundidad (DFS).

En cuestión de grafos dirigidos se puede aceverar los siguiente y es que si a el grafo se le omite la dirección de las aristas y el grafo resultante es conexo, entonces se puede asegurar que ese grafo dirigido es conexo o mejor débilmente conexo.

Por el contrario si no se omiten las dirección de las arsitas y se cumple la definición de que para culesquiera par de nodos del grafo existe un camino de u a v y de v a u entonces se dice que ese grafo es fuertemente conexo.

2.3. Hamilton

Según el profesor Eddy un camino hamiltoniano es un camino que pasa por todos los nodos sin repetir nodos, asi mismos un ciclo hamiltaniano cumple con la misma característica pero la diferencia es que el primer y ultimo nodo se repiten esto para cumplir con la caracteristica de un ciclo. Además el profesor agraga que caminos hamiltonianos es un problema no polinómicamente acotado osea NP. Ya que con la solución que se le da por medio de bactracking se tendría que montar un grafo de grafos lo que hace que la solución posea un orden de crecimiento o O(n) de $O(n!)$.[4]

Los caminos hamiltonianos poseen estas características particulares:

- Un camino sin vértices repetidos que recorre todos los vértices del grafo se llama camino hamiltoniano.
- Un camino hamiltoniano que sea un ciclo se llama ciclo hamiltoniano.
- Un grafo que tiene un ciclo hamiltoniano se llama grafo hamiltoniano.

En los grafos dirigidos, la definiciones, están pensadas para aristas dirigidas:

- Un camino dirigido en un digrafo es camino dirigido hamiltoniano si visita todos los vértices del digrafo sin repetir ninguno.
- Un ciclo dirigido hamiltoniano en un digrafo es un camino dirigido hamiltoniano que es ciclo.

- El grafo dirigido es digrafo hamiltoniano si contiene un ciclo dirigido hamiltoniano.

Para que los grafos sean hamiltonianos deben de cumplir ciertas condiciones la cuales son las siguientes:

- Un grafo hamiltoniano ha de ser conexo.
- Un grafo hamiltoniano no puede tener vértices de grado 1: en todos los vértices deben incidir al menos dos aristas, la de “entrada” y la de “salida”.
- Si S es un subconjunto del conjunto de vértices de un grafo G, escribimos G - S para designar el subgrafo que aparece al eliminar todos los vértices de S y todas las aristas adyacentes a los vértices de S.

2.4. Prim

El algortimo de Prim es un algoritmo perteneciente a la teoría de grafos, el cual sirve para encontrar el árbol recubridor mínimo en un grafo conexo y cuyas aristas están etiquetadas. Es decir que el algortimo encuentra un subcojunto de aristas que forman un árbol con todos los vétices, donde el peso total de todas las aristas en el árbol es el mínimo posible, si el grafo no es conexo entonces el algortimo encontrará el árbol recubridor mínimo para uno de los componentes conexos que forman dicho grafo. Este algortimo fue diseñado en 1930 por el matemático Vojtech Jarnik y luego de manera independiente por el científico computacional Robert C. Prim en 1957 y redescubierto por Dijkstra en 1959. Por esta razón, el algoritmo es también conocido como algoritmo DJP o algoritmo de Jarnik.[2]

Un ejemplo de pseudocódigo de este algortimo va a ser brindada en el siguiente fragmento de texto este mismo fue recuperado de la la entrada del algoritmo de Prim en Wikipedia:

1. Asociar con cada vértice v del grafo un número C[v] (el mínimo coste de conexión a v) y a un lado E[v] (el lado que provee esa conexión de mínimo coste). Para inicializar esos valores, se establecen todos los valores de C[v] a $+\infty$ (o a cualquier número más grande que el máximo tamaño de lado) y establecemos cada E[v] a un valor ”flag”(bandera) que indica que no hay ningún lado que conecte v a vértices más cercanos.
2. Inicializar un bosque vacío F y establecer Q vértices que aún no han sido incluidos en F (inicialmente, todos los vértices).
3. Repetir los siguientes pasos hasta que Q esté vacío:
   a) Encontrar y eliminar un vértice v de Q teniendo el mínimo valor de C[v]
   b) Añadir v a F y, si E[v] no tiene el valor especial de ”flag”, añadir también E[v] a F
   c) Hacer un bucle sobre los lados vw conectando v a otros vértices w. Para cada lado, si w todavía pertenece a Q y vw tiene tamaño más pequeño que C[w], realizar los siguientes pasos:
      1) Establecer C[w] al coste del lado vw
      2) Establecer E[w] apuntando al lado vw
4. Devolver F

En el caso del algoritmo de Prim se puede iniciar en cualquier vertice del grafo.

2.5. Dijkstra

El algortimo de Dijkstra o también conocido algortimo de caminos mínimos, es un algortimo para la determinación del camino más corto, dado un vértice origen, hacia el resto de los vértices en un grafo que tiene pesos en cada arista. Este algoritmo recibe su nombre gracias a Edsger Dijkstra quién describio por primera vez este algoritmo en 1959.[3]

La idea en este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices, cuando se obtiene el camino más corto desde el vértice origen hasta el resto de los vértices que componen el grafo, el algoritmo se detiene. Este algortimo no funciona con aristas de coste negativo.

"..el camino más corto en una componente conexa desde un nodo S hacia todos los demás siempre y cuando las aristas sean positivas".

La cita anterior es una pequeña definición brindada por el profesor Eddy Ramírez para el algoritmo de Dijkstra. A continuación se presenta una descripción del algoritmo recuperada de la entrada de Wikipedia del algortimo de Dijkstra:

1. Inicializar todas las distancias en D con un valor infinito relativo, ya que son desconocidas al principio, exceptuando la de x, que se debe colocar en 0, debido a que la distancia de x a x sería 0.
2. Sea a = x (Se toma a como nodo actual.)
3. Se recorren todos los nodos adyacentes de a, excepto los nodos marcados. Se les llamará nodos no marcados v_i.
4. Para el nodo actual, se calcula la distancia tentativa desde dicho nodo hasta sus vecinos con la siguiente fórmula: $dt(v_i) = D_a + d(a, v_i)$. Es decir, la distancia tentativa del nodo 'v_i' es la distancia que actualmente tiene el nodo en el vector D más la distancia desde dicho nodo 'a' (el actual) hasta el nodo vi. Si la distancia tentativa es menor que la distancia almacenada en el vector, entonces se actualiza el vector con esta distancia tentativa. Es decir, si $dt(v_i) < D_v i \rightarrow D_v i = dt(v_i)$
5. Se marca como completo el nodo a.
6. Se toma como próximo nodo actual el de menor valor en D (puede hacerse almacenando los valores en una cola de prioridad) y se regresa al paso 3, mientras existan nodos no marcados.

Una vez terminado al algoritmo, D estará completamente lleno.

2.6. Diámetro

El diámetro de un grafo son los 2 nodos cuyo camino mínimo entre elloses el máximo de todos los caminos que posea el grafo.

En Wikipedia se hace un interesante acote a el concepto de diámetro el cual es el sigueinte. Internet permite de ver desde otro enfoque la idea del diámetro: considérese por ejemplo que si se descartan los sitios que no tienen enlaces, y se escogen dos páginas web al azar, cabría preguntarse en cuántos clics se puede pasar del primer sitio al segundo. Si se supone que de cualquier sitio que enlace con otros sitios se puede llegar a cualquier otro, entonces las mayor cantidad de clics necesarios para llegar de cualquier web a otra sería el "diámetro" de la Red, vista como un grafo cuyos vértices son los sitios, y cuyas aristas son los enlaces entre los sitios.

2.7. Clique

El concepto de clique es muy sencillo pero poco conocido en nuestro país, un clique es un subgrafo completo. Para dejar claro lo que es un subgrafo completo, vamos a la definición de grafo completo la cual es si existen aristas uniendo todos los pares posibles de nodos. Es todo par a, b de nodos debe tener una arista e que los une. Se dice que un grafo completo de n vértices tiene exactamente

$$\frac{n(n-1)}{2} \tag{1}$$

El Problema del clique, que consiste en dado un grafo, decidir si existe en él un clique con un tamaño particular, es NP-completo.[5]

El término proviene de la palabra inglesa *clique*, que define a un grupo de personas que comparten intereses en común. En esta analogía, las personas serían los nodos; las relaciones de interés, las aristas; y el hecho de que todas compartan un mismo interés, el grafo completo, es decir, el clique en si.

2.8. Colorear

En Teoría de grafos, la coloración de grafos es un caso especial de etiquetado de grafos; es una asignación de etiquetas llamadas colores a elementos del grafo. De manera simple, una coloración de los vértices de un grafo tal que ningún vértice adyacente comparta el mismo color es llamado vértice coloración. Similarmente, una arista coloración asigna colores a cada arista tal que aristas adyacentes no compartan el mismo color, y una coloración de caras de un grafo plano a la asignación de un color a cada cara o región tal que caras que compartan una frontera común tengan colores diferentes.[6]

2.9. Matriz de adyacencia

El significado de la matriz es una matriz cuadrada que se utiliza como una forma de representar relaciones binarias.

Para construir una matriz de adyancencia se sigue los siguientes pasos:

1. Se crea una matriz cero, cuyas columnas y filas representan los nodos del grafo.
2. Por cada arista que une a dos nodos, se suma 1 al valor que hay actualmente en la ubicación correspondiente de la matriz.
   Si tal arista es un bucle y el grafo es no dirigido, entonces se suma 2 en vez de 1.

Existe una matriz de adyacencia única para cada grafo (sin considerar las permutaciones de filas o columnas), y viceversa. [8]

3. Descripción del Problema

Los grafos son estructuras de datos muy maleables, capaces de adaptarse a una serie enorme de problemas, mientras se cuente con datos y una relación entre ellos, se tiene por lo tanto un grafo. Los grafos tienen muchosalgoritmos asociados que son necesarios para resolver diferentes problemas.El problema presentado por el profesor fue el de crear una bilioteca de grafos en el lenguaje de programación *Prolog* con lo que se puede hacer uso del paradigma de programación lógico además de que se puede aprovechar la ventaja de que el compilador del lenguaje *Prolog* funciona como una pila, con esto para poder reducir los códigos de la mayoría de los algoritmos, ya que muchos de estos requieren de un dfs para ser solucionados.

4. Descripción de la solución

Llegar a la solución de varios de estos ejercicios fue bastante difícil ya que manejar el lenguaje de programación Prolog es complicado de entender su funcionamiento y que la verdad sabemos de que muchas de las soluciones no están dadas de la manera más optima esperada pero que al menos ahora el equipo de trabajo posee una noción de como es que este lenguaje funciona. Los ejercicios que se lograron resolver son:

- Matriz de adyacencia.
- Conexo.
- Fuertemente Conexo.
- Ciclo.
- Hamilton.
- Ciclo de Hamilton.
- Diámetro.
- Prim.

La gran ventaja es que se utilizó el método que el profesor facilitó con el que se consiguen el camino entre 2 nodos. Ya que con este método se pudo guiar el equipo de trabajo para poder sacar la solución de los otros ejercicios listados arriba.

```
camino(G,A,B,[A,B]):-member([A,B],G).
camino(G,A,B,[A|C]):-member([A,X],G), camino(G,X,B,C), \+ member(A,C),!.
```

El código de arriba es el que el profesor brindó con el que se encuentran los caminos entre 2 nodos.
El método de la matrz de adyacencia es el siguiente.

```
matriz(G,L):-quitaPeso(G,L1),flatten(L1,L2),eliminar(L2,[],L3),product(L3,L3,L4),length(L3,LE),
matriz_aux(L4,L1,LE,LE,[],[],LE,L).
```

En esta solución se puede ver que fue necesario de utilizar un método auxiliar.

```
fconexo(G,false):- quitaPeso(G,L), listanodos(G,L2), domino(L2,[X,Y]), \+ camino(L,X,Y,_),!,false.
fconexo(G,true):- quitaPeso(G,L), listanodos(G,L2), domino(L2,[X,Y]),  camino(L,X,Y,_),true,!.
```

En el método de arriba es el que se desarrollo para hacer el método de fuertemente conexo.
Esto anterior son pequeños ejemplos de los métodos desarrollados para la solución del proyecto de la biblioteca de grafos, ya que se puede notar como es que en el los métodos principales se hizo necesario el uso de funciones auxiliares para que los códigos no quedaran tan grandes.

5. Conclusiones

El lenguaje de programación *Prolog* es muy útil para darle solución a problemas de que involucren soluciones de backtracking, pero posee la desventaja de que es bastante complicado de dar las condiciones para que el método a desarrollar funcione.

Con el desarrollo de este proyecto de grafos se reforzaron muchos conceptos de teoría de grafos, ya que para el desarrollo de los métodos lo debe de estar claro es la algoritmia de los ejercicios. Con esto podemos ver la importancia de los grafos, ya que en los cursos de discretas no se toma tanto en cuenta la programación, y el curso de estructuras de datos es un curso de manejo de punteros, y con esto se puede ver el restraso de conocimiento necesario, pero con este proyecto se refuerza y crea en el estudiante el gusto por temas de ciencias de la computación.

Con este proyecto se puede ver los ejemplos de aplicaciones prácticas de los algoritmos y ejercicios desarrollados, ya que en las clases del curso el profesor pudo evidenciar estos ejemplos prácticos y dejar claro que se debe de conocer y dominar estos temas, ya que en materia de estructuras de datos es algo muy relevante.

6. Experiencia

Con respecto a las experiencias del curso podemos aceverar como grupo de trabajo que ha sido unos de los cuales se sale aprendiendo más, además de que se llega a comprender de una mejor manera el entorno en el que uno como profesional se desenvuelve, y debemos de apreciar este tipo de cursos ya que en Universidades privadas no poseen en su malla de estudio un curso en el que se comprende lo que son lenguajes, gramáticas y la importancia que estas poseen para nosotros futuros profesionales en informática. Además citamos a el profesor Eddy:

> "si hasta la gente que estudia diseño estudia lo que son gramáticas, por qué nosotros que somos de informática no lo hariamos"

Así que bien además algo que destacamos del profesor la verdad no tenemos la certeza si los demás profesores lo hagan, y es que para él es relevante dar el contexto histórico de los lenguajes o algoritmos que vayan a explicar y como esto es una retrospectiva visto en términos de los objetivos del curso, destacamos eso en el profesor; aunque estemos tomando este espacio para poder escribir de forma más libre, y expresar más nuestro sentir a título personal.

Queremos agradecerle a el profesor ya que aunque con los proyectos, llegamos a tener dificultades, dimos nuestro mayor esfuerzo, pero sobre todo recibimos conocimiento que ahora nadie nos va a poder quitar, ya que con el primer proyecto de programación genética, recibimos una teoría que es bastante valiosa, y que aunque el objetivo era evaluar la programación funcional, los conceptos adicionales que recibimos, estamos seguros de que nos serán de gran utilidad en el futuro. Y ahora con esta biblioteca de grafos, con la que tuvimos la oportunidad de repasar conceptos con los cuales nos habiamos enfrentado anteriormente y tal vez olvidamos, o peor aún solo los aprendimos para una evaluación y después de la misma los descartamos, sin ver lo importantes y relevantes que son, sin olvidar que además se logró un entendimiento básico del paradigma lógico, objetivo del curso.

Referencias

[1] *Clase de Paragidmas*, 2019.

[2] Wikipedia. Algoritmo de prim.

[3] Wikipedia. Algoritmo de dijkstra, 2019.

[4] Wikipedia. Camino hamiltoniano, 2019.

[5] Wikipedia. Clique, 2019.

[6] Wikipedia. Coloración de grafos, 2019.

[7] Wikipedia. Grafo, 2019.

[8] Wikipedia. Matriz de adyacencia, 2019.

[9] Wikipedia. Teoría de grafos, 2019.