Guión Tema 4. GRAFOS

Explicación y conceptos importantes sobre tema de grafos. Iré relacionando las prácticas con la teoría para que os resulte más sencillo comenzar las prácticas de grafos. Es importante que hayáis leído y tengáis a mano el fichero con la presentación (T4. Grafos.pdf) pues en él se explican los conceptos de los que aquí hablo y que no voy a repetir. También es importante ir leyendo e implementando los ejercicios del fichero de prácticas (T4.PrácticasGrafos.pdf). La intención es ayudaros a comprender mejor el tema. Es posible que alguno de vosotros ya lo hayáis comprendido y no necesitéis este documento. Para no aburriros mucho solo he incluido, de momento, la parte correspondiente a la tarea2 que os programé la semana pasada. Quienes la hayan hecho creo que no tendrán problema en seguirme, quienes todavía no hayan empezado deberían ir empezando, pues al finalizar el tema, se pedirá alguna tarea.

1. Nivel de abstracción o definición.

Teoría. En este apartado debe quedar claro que un **grafo** queda completamente definido por su conjunto de vértices y el conjunto de aristas que los conectan. El **orden** de un grafo es su número de vértices.
Una **arista** queda determinada por el par de vértices que une, (v,w). Esto da lugar al concepto de **adyacencia** entre vértices, este concepto es muy importante porque en él se basará la representación de grafos en memoria. También da lugar a la primera clasificación de **grafos en dirigidos y no dirigidos**, dependiendo de si importa o no el orden del par de vértices en las aristas. Si las aristas del grafo tienen asociado un tercer valor (peso o coste) hablaremos de **grafos ponderados**. Aunque en estos primeros apartados trabajamos con grafos no ponderados.
En principio se presentan grafos en general, pero los que nos interesan son los **grafos simples** que se caracterizan porque no hay más de una arista conectando un par de vértices, y no existen aristas de un vértice a sí mismo (bucle).
La definición de **camino simple** también es importante pues algunos de los algoritmos que veremos lo que buscan precisamente es encontrar caminos simples mínimos.
Para finalizar este apartado queda el concepto de grado de un vértice, que también vamos a utilizar en algún algoritmo.

Prácticas. Este apartado es teórico, simplemente hay que tener claros los conceptos que se utilizarán en las prácticas.

2. Nivel de Representación.

Teoría. Una vez que hemos decidido como queda determinado un grafo (por su conjunto de vértices y aristas) debéis centraros en su **representación mediante listas de adyacencia** (página 16), que es la que utilizaremos en prácticas. Si entendéis bien esta representación, el resto será fácil. Tened en cuenta que necesitamos dos estructuras, una para representar los vértices y otra para representar las aristas o arcos. Los vértices se representan en el **directorio de vértices** y las aristas mediante **listas enlazadas,** que se denominan **listas de adyacencia** pues contienen para cada vértice todos sus vértices adyacentes. Para reunir todas estas listas se utiliza el directorio de vértices, un vector donde los índices representan los vértices, y el contenido de cada celda, de momento, un puntero a su lista de adyacencia. Este contenido lo ampliaremos a lo largo del tema para ir almacenado en el vector más información de los vértices. Para definir el Grafo sólo es necesario añadir al directorio vértices, un entero que indique el orden del grafo (número de vértices que tiene).
Los grafos dirigidos y no dirigidos se representan igual, la diferencia está en el contenido de las listas de adyacencia de cada vértice:
- en grafos dirigidos la arista (v, w) indica que w es adyacente a v, y por tanto w estará en la lista de adyacencia de v
- en los grafos no dirigidos no importa el orden en las aristas y por tanto la arista (v,w) indica que v es adyacente a w (v estará en la lista de adyacencia de w) y que w es adyacente a v(w estará en la lista de adyacencia de v).

Prácticas. La parte de representación ya está hecha en "**grafos.h**". Tened en cuenta que el tipoVértice del que hablamos en teoría se identifica aquí por "**`vertices`**", en esta estructura ya se han añadido todos los campos de información que necesitaremos a los largo del tema. Por eso el directorio de vértices no es un vector tan sencillo como el que introducimos en este apartado en teoría.
La primera función del **ejercicio 1** (**creaGrafo28**) tienen mucho que ver con este apartado y es importante entenderla, aunque está hecha no la paséis por alto. Entender la representación del grafo en memoria es básico para entender el resto de algoritmos. Podéis hacer otras funciones que creen otros grafos inventados o los que aparecen en teoría.

3. Recorridos en grafos

Teoría. Existen, como en árboles, dos tipos básicos de recorridos: **amplitud** y **profundidad**. La principal diferencia es que, en árboles existe un camino único desde la raíz (nodo especial) a cualquier nodo y en grafos (no distinguimos un vértice especial del resto) tomando un vértice inicial podemos encontrarnos con otro vértice varias veces en el recorrido. Es por eso que, cada vez que se visite un vértice, hay que marcarlo para no volver a visitarlo si lo encontramos otra vez en el recorrido. Esto da lugar a la primera modificación de las declaraciones básicas, además de conocer la lista de adyacencia de cada vértice, se necesita saber si ya ha sido visitado. Por tanto el directorio de vértices pasa de ser un vector cuyo contenido son punteros a listas de adyacencia a un vector cuyas celdas contienen información de tipoVértice (página 22). Deteneros en la definición de tipoVértice, es una estructura con dos campos **alanzado** (indicará si el vértice ha sido visitado) y **listaAdyacencia** con un enlace a la lista de cada vértice. Observad que entre estos dos campos hay "puntos suspensivos", porque iremos incluyendo los que necesitemos según vayamos avanzando en el tema.
Como no hay un vértice especial, los recorridos necesitan como parámetro de entrada un vértice inicial. Además debe quedar claro que, dado un grafo, los recorridos pueden ser diferentes, incluso aunque el vértice de inicio sea el mismo. Diferentes en el sentido de que el orden en que se visitan los vértices puede ser ligeramente diferente.

Prácticas. La segunda función (**iniciar**) del **ejercicio1** es muy importante pues se va a utilizar en el resto de algoritmos. De momento, nos basta con inicializar el campo alcanzado del directorio de vértices a FALSO (ningún vértice ha sido visitado) para poder aplicar los recorridos correctamente. Sin embargo, vamos ya a inicializarlos todos para tener preparada esta función para el resto del tema: los campos **`ordenTop`** y **`anterior`** con valor valor 0, los campos **`distancia`** y **`coste`** con valor INF (constante definida en grafos.h con valor 999999 que representa para nosotros infinito), el campo **`gradoEntrada`** tienen que contener el grado de entrada de cada vértice y para calcularlo hay que recorrer las listas de adyacencia (¡cuidado con la eficiencia!). Por último el campo **`lista`** no se toca, representa las listas de adyacencia que contienen los vértices adyacentes de cada vértice del grafo. ¡¡ Este campo nunca se inicia pues nos quedamos sin aristas y por tanto sin grafo!!!
Las funciones de los recorridos, aunque están hechas, hay que mejorarlas pues como veréis si no lo hacéis no os saldrán los resultados esperados que se muestran en los ficheros de resultados (*.txt)

4. Ordenación Topológica

Teoría. Teniendo en cuenta en que consiste (clasificación de los vértices de un grafo dirigido acíclico tal que si existe un camino de v a w , v aparece antes que w en la clasificación) debe quedar claro que:
- solo se puede aplicar a grafos dirigidos y acíclicos (aunque nos servirá para detectar si un grafo es cíclico)
- los primeros en clasificarse deben ser los vértices cuyo grado de entrada sea 0 (a ellos no llega ninguna arista, no hay ningún camino que llegue a ellos)
- cada vez que se clasifique un vértice, si se decrementa el grado de entrada de sus vértices adyacentes, los siguientes en clasificarse serán los que alcancen primero su grado de entrada a cero (esto supone que ya se han clasificado todos los vértices que tienen un camino hacía él)

Necesitamos para este algoritmo la información sobre el grado de Entrada de cada vértice, para ello incluimos en la estructura tipoVértice el campo gradoEntrada. Además, para saber el orden topológico que se le asigna a cada vértice, se incluye otro campo denominado ordenTopológico.
A partir de este algoritmo, todos los algoritmos con grafos tienen dos versiones, una inicial y otra mejorada. En algunos casos, la inicial se mejora con una cola básica, pero como veremos … en otros casos se necesita una cola de prioridad (montículo binario).
Si entendéis bien las dos versiones de este algoritmo, el resto de algoritmos tienen la misma idea básica.

Prácticas. El **ejercicio 2** pide la implementación de las dos versiones de ordenación topológica, con una pequeña modificación que permita identificar si el grafo es cíclico. ¡Si es cíclico no se puede establecer la ordenación topológica!

La semana que viene seguiré con el resto del tema.