Algorithms for Problem Solving – 11650

Recorrido de grafos

Jon Ander Gómez Adrián
jon@dsic.upv.es

Departament de Sistemes Informàtics i Computació
Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

7 de marzo de 2014

Tipos de Grafos

A la hora de resolver problemas de grafos, o problemas en cuya solución podamos utilizar grafos, debemos escoger la estructura de datos más adecuada, y para ello es necesario considerar varias propiedades de éstos:

- Dirigidos/No dirigidos
- Ponderados/No ponderados
- Con ciclos/Acíclicos
- Simples/Con arcos duplicados/Con arcos reflexivos
- *Embedded* (con posiciones en 2D)/Topológico
- Implícito/Explícito
- Etiquetado/No etiquetado

Estructuras para Grafos I – Matriz de adyacencias

- Si el grafo tiene n nodos los arcos se representan mediante una matriz de $n \times n$.
- Es el modo más rápido para saber si dos nodos cualesquiera están conectados.
- También permite la inserción y borrado de arcos de manera inmediata.
- El consumo de memoria puede ser excesivo. Especialmente en grafos con muchos nodos pero poco densos (bajo grado de conectividad).

Estructuras para Grafos II – Listas de arcos

- Cada nodo contiene una lista enlazada de punteros a los nodos adyacentes.
- Es de particular interés para grafos dispersos o poco densos.
- Utiliza la memoria necesaria a cambio de que saber si dos nodos están conectados requiere una búsqueda secuencial (coste lineal) en la correspondiente lista.
- Para un grafo no dirigido si dos nodos A y B son adyacentes debe existir un arco en la lista de A que apunte a B y viceversa.
- Para algunos algoritmos de búsqueda el recorrido de la lista de nodos adyacentes de un nodo se realiza de manera eficiente.

Estructuras para Grafos III – Listas de arcos en vectores

- Se estructura la información de igual manera que para listas enlazadas, pero por cada nodo se dispone de un array con las referencias a los nodos adyacentes.
- Esta información se puede almacenar en una única matriz con tantas filas como nodos tenga el grafo y un máximo de nodos adyacentes por cada nodo que denominaremos grado máximo.
- Por cada nodo se debe mantener un contador de arcos o nodos adyacentes.

Estructuras para Grafos IV – Lista única de arcos

- Una simplificación más puede ser disponer de una única lista de arcos.
- Esta organización tiene un coste más alto para responder a la pregunta de si dos nodos están enlazados.
- Sin embargo resulta mejor para ciertos algoritmos, como el de Kruskal para obtener árboles de expansión mínima (*minimum spanning trees*).

BFS: Recorrido o búsqueda en anchura

- Es la solución para encontrar el camino más corto en un grafo no ponderado (todos los arcos tienen el mismo peso).
- El orden en que son explorados los nodos desde el origen viene dado por el número de saltos para alcanzar a cada nodo.
- Este algoritmo se basa en una cola FIFO de donde se van sacando los nodos a procesar.
- Por cada nodo se añaden a la cola todos sus adyacentes que todavía no estén en la cola ni hayan sido procesados.
- Un nodo se considera procesado cuando todos sus adyacentes han sido visitados y añadidos a la cola si todavía no lo habían sido.

DFS: Recorrido o búsqueda en profundidad

- La búsqueda en profundidad (DFS) puede verse como la de anchura (BFS) cambiando la cola (FIFO) por una pila (LIFO).
- Sin embargo, al igual que en *backtraking* resulta más cómodo utilizar un diseño recursivo del algoritmo.
- La exploración por DFS de un grafo o de un árbol (caso particular de grafos acíclicos y dirigidos) puede realizarse en tres modalidades diferentes: interno, preorden o postorden, dependiendo de cuando se decide procesar un nodo.
- Mediante DFS es fácil encontrar ciclos (bucles) si durante la exploración nos encontramos que podemos alcanzar de nuevo un nodo que ya ha sido explorado.

Componentes conexas

- Una componente conexa de un grafo no dirigido es un subconjunto de nodos en el que cualesquiera dos nodos están conectados mediante un camino.
- En un grafo pueden estar todos los nodos conectados o estar formado por varias componentes conexas.
- Las componentes conexas se pueden detectar mediante DFS o BFS.
- Una vez aplicado el algoritmo desde un nodo cualquiera, si quedan nodos por explorar eligiremos uno de ellos y volveremos a aplicar el algoritmo hasta que hayan sido explorado todos.
- Cada vez que lo apliquemos obtendremos una componente conexa.

Orden topológico

- La ordenación topológica es de especial interés en DAG's.
- Permite obtener un orden de procesamiento de izquierda a derecha de manera que un nodo no sea procesado después de ninguno de su sucesores.
- La estrategia es fácil, al ser un grafo dirigido el primer paso a realizar es calcular el grado de entrada (número arcos que lo apuntan) para cada nodo.

  Aquellos con grado de entrada cero se añaden a una cola FIFO.
- Cuando se procesa un nodo se resta uno al grado de entrada de todos sus adyacentes. Aquellos cuyo grado queda a cero se añaden a la cola.
- Si el algoritmo para porque la cola se vacía ha acabado satisfactoriamente, si para porque el número de nodos procesados supera el número de nodos en el grafo es por culpa de un ciclo.

Bicoloring

110901/10004

- Aplicaremos *Breadth-First Search*.
- Inicialmente todos los nodos marcados sin color.
- Escogemos uno al azar y lo marcamos de rojo o de negro.
- Cada vez que procesemos un nodo, todos sus vecinos por colorear los marcaremos con el color alternativo y los añadiremos a la cola.
- Los vecinos que ya estén coloreados comprobaremos que tienen el color alternativo al nodo actual, de no ser así ya podemos finalizar, no es *bi-colorable*.

Playing With Wheels

110902/10067

- Es es un claro ejemplo de grafo *implícito* que no construiremos. Su número de nodos es 10000 menos los prohibidos.
- Los nodos alcanzables desde un nodo cualquiera, máximo 8, los obtendremos cuando lo procesemos.
- A partir de una combinación generaremos las 8 posibles y pondremos en la cola las que no estén prohibidas.
- La estrategia también es BFS, y aunque no construyamos el grafo completo necesitamos marcar los visitados para evitar volver a pasar por ellos.

110903/10099

- Estrategia estilo Dijkstra para buscar el camino más corto.
- Actualizar el caudal máximo de cada camino posible que llegue al destino.
- Ojo que el caudal máximo de cada camino es el mínimo de todos los arcos que lo componen.
- Si alcanzamos un nodo cuyo caudal con un camino que lo alcanzó antes es menor que el caudal con que lo alcanzamos ahora, entonces le actualizamos el caudal.
- En otras palabras, el mejor camino desde el nodo inicial a un nodo cualquiera no es el más corto, sino el de más caudal.

Resumen problemas

Del tema 9 hemos abordado los problemas:

- 110901/10004 "Bicoloring"
- 110902/10067 "Playing With Wheels"
- 110903/10099 "The Tourist Guide"