Algorísmia

Lliurament: Resolució del problema 29.
Data: Setmana del 31/03, 2015-2016 Q2.
Nom: Ricard Meyerhofer Parra.

Introducción al problema

El problema que se nos pide solucionar es el de dado un grafo G, hallar el subconjunto de vértices que tienen grado mayor o igual a k. Así pues lo que se nos pide es dar el **k-core** (subgrafo máximo conexo de G en el cual todos los vértices v tienen como mínimo grado k) de G.

Para ejemplificar lo que es un k-core, se adjunta el siguiente ejemplo donde se puede apreciar que no viene determinado por el grado del Grafo inicial. Por ejemplo, el 3-core no cuenta con el nodo amarillo de la parte central derecha porque el nodo azul no tiene grado >= 3 y como consecuencia de ello, tiene grado 2 que hace que no pertenezca a un 3-core.

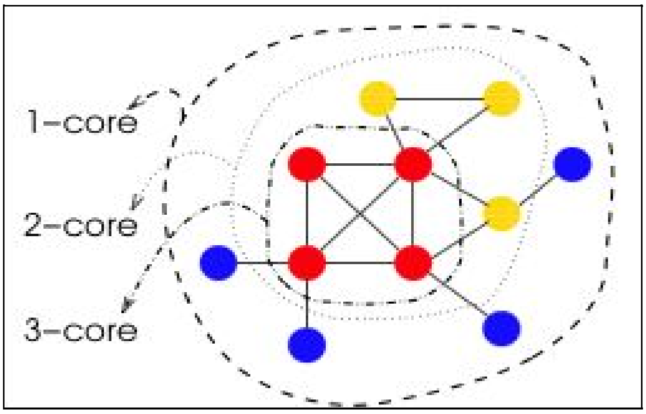

Imagen 1: Ejemplo de un k-core.

Una vez sabemos ya a qué problema nos enfrentamos y hemos explicado lo que es un k-core, buscamos una estrategia para solucionarlo.

Estrategia aplicada

Cómo hallaremos los k-cores? Hallaremos los k-cores simplemente eliminando cualquier vértice que no tenga grado mayor o igual a k. Finalmente de las componentes conexas resultantes, seleccionaremos la mayor de ellas (en caso de que haya varias)

Con mayor detalle, la estrategia aplicada hará lo siguiente:

- Recorrer el grafo y anotar el grado de cada vértice **O(|V|)**, si el grado de cada vértice es >= k y es una sola componente ya tenemos la solución, si hay más de una componente, la mayor de ellas es el resultado (podemos saber esto con coste **O(|V|+|E|)**.
- Ir a través de los vértices en busca de vértices con grado menor que k. Cuando encontramos uno que no tiene el grado adecuado, lo quitamos del grafo y actualizamos el grado de sus vecinos (también eliminamos los vecinos que pasan a tener grado menor a k).

  Se necesita mirar cada vértice como mínimo una vez (coste **O(|V|)**) y actualizar los grados como máximo una vez para cada arista (**O(|E|)**), lo que da un coste total de **O(|V|+|E|)**.
- Como no sabemos si tenemos componentes conexas, tal y como se explica en las transparencias de la asignatura, aplicamos DFS y contamos cuántas veces explore es llamado. Cada vez que el DFS llama a este, nos dice a qué componente conexa pertenece (coste total del algoritmo **(O(|V|+|E|)**).
- Si no hay componentes conexas nuestro resultado es el k-core. Si hay más de una componente conexa, nos quedaremos con la mayor.

Así pues nuestra estrategia tendrá un coste de **O(|V|+|E|)**.

<u>Estructura de datos usada:</u>

- **Vector de grado de cada vértice** donde se almacenarán los grados de los vértices y se irán actualizando para así no tener que recalcular estos.
- **Grafo implementado con lista de adyacencias** por ejemplo.

Código

k-core (pseudocode)

```
vector<<vector<int> > Graph;

void eliminaGraf(Graph g, int posicio, int k) {
   for (int j = 0; j < g[posicio].size(); ++j){ //cost O(E)
         --degree[g[j]]; //decrementem els graus
         if (degree[g[j]] < k) { //si els veins es veuen afectats
             eliminaGraf(g,g[j], k); //recursivament fem el mateix
            g[g[j]].erase();
          }
   }
}

Graph kCore(Graph g, int k) {
   vector<int> degree (g.size(),0);
   boolean kcore = true;
   // Creem un vector amb el grau de cada vertex.
   for (int i = 0; i < g.size(); ++i) {
         int grau = g[i].size();
         degree[i].push_back(grau);
         if (grau < k) kcore = false;
   }
   if (kcore) return componentConexamajor(g);
   else {
         for (int i = 0; i < degree.size(); ++i){ //cost O(V)
                  if (degree[i] < k) {
                       eliminaGraf(g,g[i],k); //actualitza graus i aplica als veins
                       g[i].erase();
                   }
         }

   }
   return componentConexamajor(g);
}
```