graph - How to find maximum induced subgraph H of G such that each vertex in H has degree ≥ k

Here is an excise for graph.

> Given an undirected graph G with n vertices and m edges, and an integer k, give an O(m + n) algorithm that finds the maximum induced subgraph H of G such that each vertex in H has degree ≥ k, or prove that no such graph exists. An induced subgraph F = (U, R) of a graph G = (V, E) is a subset of U of the vertices V of G, and all edges R of G such that both vertices of each edge are in U.

My initial idea is like this:

First, this excise actually asks that we have all vertices S whose degrees are bigger than or equal to k, then we remove vertices in S who don't have any edge connected to others. Then the refined S is H, in which all vertices have degree >= k and the edges between them is R.

In addition, it asks O(m+n), so I think I need to a BFS or DFS. Then I get stuck.

In BFS, I can know the degree of a vertex. But once I get the degree of v (a vertex), I don't know other connected vertices except for its parent. But if the parent doesn't have degree >= k, I can't eliminate v as it may still be connected with others.

Any hints?

Edit:

According to the answer of @Michael J. Barber, I implemented it and update the code here:

Can anyone have a look at the key method of the codes `public Graph kCore(Graph g, int k)` ? Do I do it right? Is it O(m+n)?

```
class EdgeNode {
   EdgeNode next;
   int y;
}

public class Graph {
   public EdgeNode[] edges;
   public int numVertices;

   public boolean directed;

   public Graph(int _numVertices, boolean _directed) {
      numVertices = _numVertices;
      directed = _directed;
      edges = new EdgeNode[numVertices];
   }

   public void insertEdge(int x, int y) {
      insertEdge(x, y, directed);
   }

   public void insertEdge(int x, int y, boolean _directed) {
      EdgeNode edge = new EdgeNode();
      edge.y = y;
      edge.next = edges[x];
      edges[x] = edge;
      if (!_directed)
          insertEdge(y, x, true);
   }

   public Graph kCore(Graph g, int k) {
      int[] degree = new int[g.numVertices];
      boolean[] deleted = new boolean[g.numVertices];
      int numDeleted = 0;
      updateAllDegree(g, degree);// get all degree info for every vertex

      for (int i = 0;i < g.numVertices;i++) {
          **if (!deleted[i] && degree[i] < k) {
           deleteVertex(p.y, deleted, g);
          }**
      }

      //Construct the kCore subgraph
      Graph h = new Graph(g.numVertices - numDeleted, false);
      for (int i = 0;i < g.numVertices;i++) {
         if (!deleted[i]) {
          EdgeNode p = g[i];
          while(p!=null) {
             if (!deleted[p.y])
                h.insertEdge(i, p.y, true); // I just insert the good edge as directed,
because i->p.y is inserted and later p.y->i will be inserted too in this loop.
                p = p.next;
             }
          }
       }
      }

      return h;
  }

  **private void deleteVertex(int i, boolean[] deleted, Graph g) {
     deleted[i] = true;
     EdgeNode p = g[i];
```

```
      while(p!=null) {
         if (!deleted[p.y] && degree[p.y] < k)
            deleteVertex(p.y, deleted, g);
         p = p.next;
      }
   }**

   private void updateAllDegree(Graph g, int[] degree) {
      for(int i = 0;i < g.numVertices;i++) {
         EdgeNode p = g[i];
         while(p!=null) {
            degree[i] += 1;
            p = p.next;
         }
      }
   }

}
```

algorithm data-structures graph subgraph

edited Apr 18 '12 at 17:15 asked Apr 18 '12 at 7:53
Jackson Tale
10k 19 86 189

I am trying to use the following approach. 1) Use BFS/DFS to find out all vertices of degree >=k. Now these vertices will surely will be in the induced subgraph. 2) Now we need to find to find out whether any of these are connected to each other. We can go through the adjacency list for each of these vertices. O (m). Overall O(m+n) + O(m)= O(m+n). Is the algorthm logic correct? – Sumit Trehan May 21 '14 at 17:38

1 Answer

A maximal induced subgraph where the vertices have minimum degree *k* is called a *k*-core. You can find the *k*-cores just by repeatedly removing any vertices with degree less than *k*.

In practice, you first evaluate the degrees of all the vertices, which is O(m). You then go through the vertices looking for vertices with degree less than *k*. When you find such a vertex, cut it from the graph and update the degrees of the neighbors, also deleting any neighbors whose degrees drop below *k*. You need to look at each vertex at least once (so doable in O(n)) and update degrees at most once for each edge (so doable in O(m)), giving a total asymptotic bound of O(m+n).

The remaining connected components are the *k*-cores. Find the biggest one by evaluating their sizes.

edited Apr 18 '12 at 11:14 answered Apr 18 '12 at 8:09
Michael J. Barber
17k 6 42 67

Thanks. How about we do like this. First I clone G to H, i.e., H is exactly the same as G in the beginning. Then I a BFS. The reason I chose BFS is because I can calculate the degree of each vertex and process the edges in one go, (m+N). Each time I meet a vertex whose degree is less than k, I just delete it and all related edges in H. However, when deleting vertices and associated edges in adjacency list is bothering. – Jackson Tale Apr 18 '12 at 10:42

could you please have a look at my codes? – Jackson Tale Apr 18 '12 at 11:33

@JacksonTale You can iterate through the vertices in any order you like; BFS is fine, but remember the graph may not be connected. Whenever you reduce the degree of a vertex, you also need to remove any vertices whose degree drops below k; you'll need to track the degrees - recalculating the degrees would lead to O(m^2). Interleaving the initial degree calculation with the processing won't affect the asymptotic performance, so I would do it in two steps, which sounds easier. Finally, you don't need to duplicate the graph; all you really need is an array of degrees. – Michael J. Barber Apr 18 '12 at 11:38

@JacksonTale If I understand your program correctly, you've missed the possibility that deleting a vertex could invalidate a vertex you've already visited. Whenever you reduce the degree for a vertex, you need to test again. You can't just test when constructing the subgraph: leaving it out then might invalidate other vertices. – Michael J. Barber Apr 18 '12 at 11:52

Yeah, you are right. I will refine my code – Jackson Tale Apr 18 '12 at 13:19