Finding the nodes that have degree at least 3 in an undirected graph

I want to come up with a good algorithm that takes as input an undirected graph G and checks which vertices to include in an output graph G', such that each vertex in G' has degree at least 3.

I've thought about traversing the graph once to eliminate any vertex with degree less than 3 since none of these can be included in the output. Then, I could have a set S that initially contains some arbitrary vertex that hasn't been eliminated; the rest of the vertices are kept in the set T.

If I keep with this idea, I'm not sure how to start moving in vertices from T to S and at which point to delete them. I also wonder if there's a way to use BFS to scan and see if each vertex has degree at least three in the output tree of the BFS procedure. Any pointers/ideas would be appreciated; thanks!

algorithms graphs graph-theory algorithm-analysis search-algorithms

edited Mar 10 at 13:12
Mario Cervera
1,647 1 5 16

asked Mar 10 at 3:02
user67544
16 1

2 Answers

This problem is known as a maximum k- core (in your case $k = 3$) and can be solved in polytime. The idea is the one you mentioned : simply delete vertex with degree less than 3 until you don't have such vertices.

More accurate description of the algorithm:

Store your graph as adjacency list. Whenever you find a vertex with degree less than k mark it as deleted and then remove it from neighbors list of its neighbors. In the end you can shift indices if you really need them to be in consecutive order.

edited Mar 10 at 5:06

answered Mar 10 at 3:41
Eugene
857 1 1 10

As mentioned in Eugene's answer, you are looking for a 3-core of the graph $G = (V, E)$. If the graph is sufficiently small, you can find any k-core in linear time using a bounded-height priority queue. This kind of priority queue is typically implemented as an array of buckets, where any given bucket at index i holds all of the items that have priority i. Thus, if you consider the items to be the vertices of G and their priorities to be their respective degrees, then this data structure allows you to keep track of the vertices of G sorted by degree.

The algorithm to find a k-core is as follows:

1. Initialize the bounded-height priority queue. This data structure supports insertions in $O(1)$ time. Therefore, if you haven't precalculated the degrees of the vertices of G, this step takes $O(|V| + |E|)$ time.
2. Find the vertex v of minimum priority (that is, of minimum degree). This operation takes $O(1)$ time.
3. Remove v and subtract 1 from the degrees of all the neighbours of v. When the degree of a vertex is modified, the vertex must be reallocated within the queue; nonetheless, this operation takes $O(1)$ time because the bounded-height priority queue is an array-based data structure that is indexed by degree.
4. Go to step 2 until there is no vertex v such that $d(v) < k$, where $d(v)$ is the degree of v.

Observe that the above greedy algorithm takes linear time overall because processing each vertex takes time proportional to its degree and the sum of all vertex degrees is linear in $|E|$.

edited Apr 13 at 12:48
Community ♦
1

answered Mar 10 at 9:47
Mario Cervera
1,647 1 5 16

Interestingly the Wikipedia article says that it is a maximal vertex set such that every degree is at least k and proposes equivalence that you take a graph and remove small degrees. I'd say it is rather pretty imprecise. The definition we use in graph theory is that k-core is an induced subgraph in which degrees are at least k. The maximum k-core of the graph then can be found in linear time. However, the minimum k-core is $\mathcal{NP}$-hard problem. – Eugene Mar 12 at 11:35