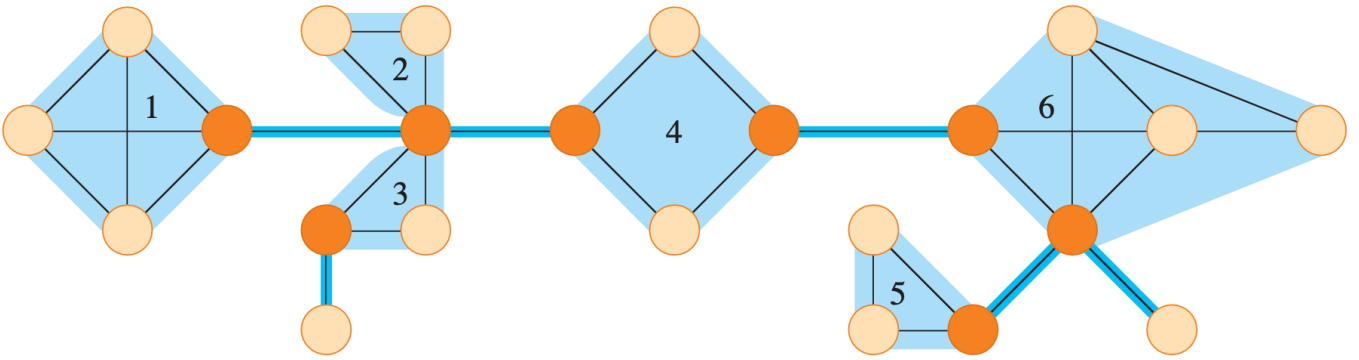


## 1 CLRS 20-2

Let  $G$  be a connected, undirected graph. An **articulation point** of  $G$  is a vertex whose removal disconnects  $G$ . A **bridge** of  $G$  is an edge whose removal disconnects  $G$ . A **biconnected component** of  $G$  is a maximal set of edges such that any two edges in the set lie on a common simple cycle. Figure 20.10 illustrates these definitions. You can determine articulation points, bridges, and biconnected components using depth-first search. Let  $G_\pi = (V, E_\pi)$  be a depth-first tree of  $G$ .



**Figure 1** The articulation points, bridges, and biconnected components of a connected, undirected graph for use in Problem 20-2. The articulation points are the orange vertices, the bridges are the dark blue edges, and the biconnected components are the edges in the light blue regions, with a *bcc* numbering shown.

1. Prove that the root of  $G_\pi$  is an articulation point of  $G$  if and only if it has at least two children in  $G_\pi$ .
2. Let  $v$  be a nonroot vertex of  $G_\pi$ . Prove that  $v$  is an articulation point of  $G$  if and only if  $v$  has a child  $s$  such that there is no back edge from  $s$  or any descendant of  $s$  to a proper ancestor of  $v$ .
3. Let

$$v.\text{low} = \min \begin{cases} v.d, \\ w.d : (u, w) \text{ is a back edge for some descendant } u \text{ of } v \end{cases}$$

Show how to compute  $v.\text{low}$  for all vertices  $v \in V$  in  $O(E)$  time.

4. Show how to compute all articulation points in  $O(E)$  time.
5. Prove that an edge of  $G$  is a bridge if and only if it does not lie on any simple cycle of  $G$ .
6. Show how to compute all the bridges of  $G$  in  $O(E)$  time.
7. Prove that the biconnected components of  $G$  partition the nonbridge edges of  $G$ .

8. Give an  $O(E)$ -time algorithm to label each edge  $e$  of  $G$  with a positive integer  $e.bcc$  such that  $e.bcc = e'.bcc$  if and only if  $e$  and  $e'$  belong to the same biconnected component.

## 2 Analyze topological sort

An alternative algorithm for topological sort was given at our course without correctness proof. Can you prove the correctness of this algorithm using the properties of source and sink? And try to analyze the time complexity of this algorithm.

### Alternative Algorithm for Topological Sort

- (1) Find a source node  $s$  in the (remaining) graph, output it.
- (2) Delete  $s$  and all its outgoing edges from the graph.
- (3) Repeat until the graph is empty.

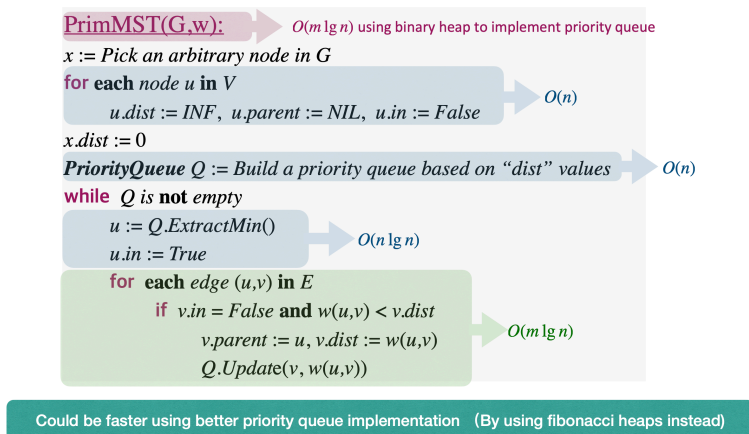
Formal proof of correctness?  
How efficient can you implement it?

**Figure 2** The alternative algorithm for topological sort in slide 14 *Application of DFS*

## 3 \*Implement Prim's Algorithm with Fibonacci Heap

*Note: This question is optional.*

### Prim's Algorithm



**Figure 3** Prim's algorithm in slide 15 *Minimum Spanning Tree*

We've mentioned that Prim's Algorithm can be implemented with a priority queue. Try to implement Prim's Algorithm with Fibonacci Heaps <sup>1</sup>.

1. Since we don't teach Fibonacci Heaps in our class, you should learn it on your own. [↗](#)