Introduction to Biconnectivity

Special class

Sidhant Bansal • Nov 20, 2020

Intro To Biconnectivity

- Sidhant Bansal
(Some of the course content used has been created by Tanuj Khattar in his blog-post and lecture video)

Agenda

1. Terminology
    a. Articulation Point
    b. Bridges
    c. Bridge component
    d. Biconnected component
2. How to implement bridge finding?
3. Bridge Tree
    a. Definition
    b. Examples
    c. Properties + Proofs
    d. Implementation
4. Problems
    a. Easy
    b. Hard

What are articulation points?
What are bridges?

1. **Bridge edge :** A bridge edge in an undirected graph is an edge whose removal increases the number of connected components in the graph by 1. (For more info [Bridges in a graph - GeeksforGeeks](#))

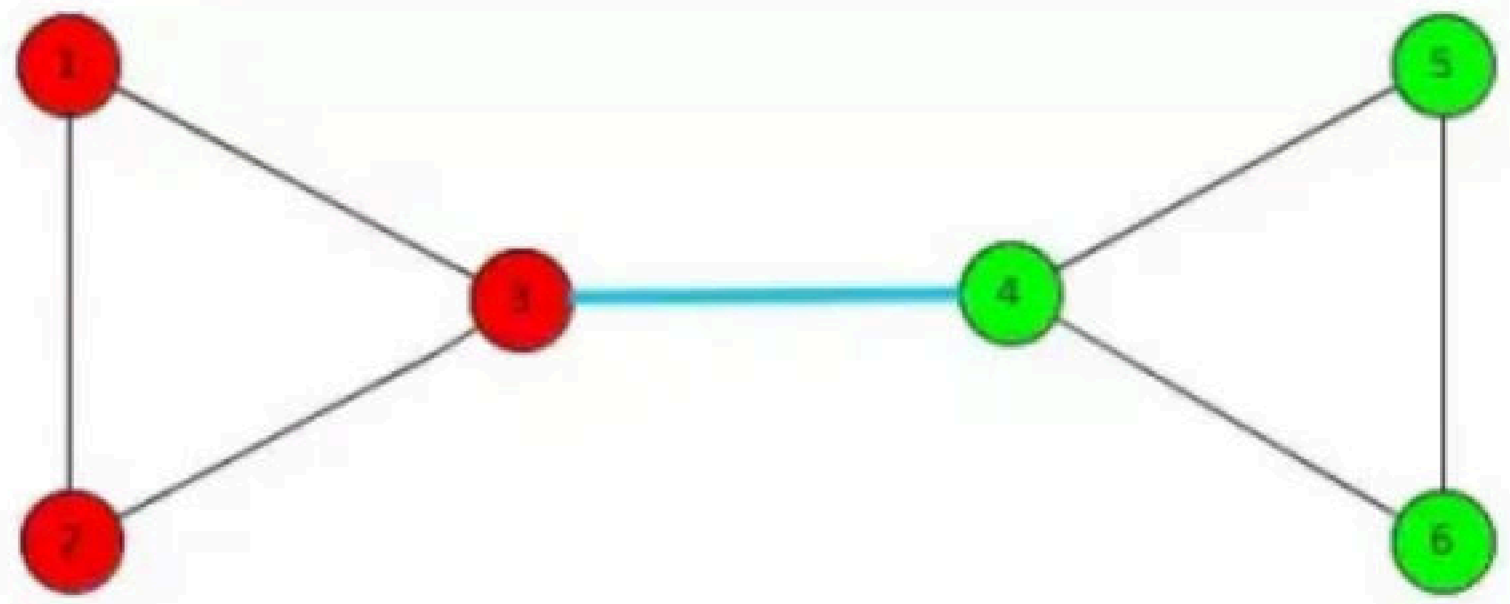
2. **Articulation Points / Cut Vertices :** An articulation point in an undirected graph is a vertex whose removal (and corresponding removal of all the edges incident on that vertex) increases the no of connected components in the graph by at-least 1. (For more info [Articulation Points (or Cut Vertices) in a Graph - GeeksforGeeks](#)).

Examples.

What is a biconnected component?
What is a bridge component?

1. **Biconnected Components :** A biconnected component of a given graph is the maximal connected subgraph which does not contain any articulation vertices. (For more info [Biconnected components])

1. **Bridge Component :** A bridge component of a given graph is the maximal connected subgraph which does not contain any bridge edges. eg :



Poll Question

Can a bridge component have an articulation point inside it? (True / False)

We will focus on Bridges and Bridge Components in this lecture

How do we find bridges?

1. Slow approach: O(E(E + V))
2. Fast approach: O(V + E)
   a. Root arbitrarily
   b. Run DFS (keep track of discovery time of each node in **disc[]**)
   c. For each node also calculate **min(discovery time)** (in **low[]**)
   d. If **low[v]** > **disc[u]** (for an edge in DFS tree going from u to v) then **u to v edge is a bridge**

Popularly known as Tarjan's Algorithm (can be modified to find Articulation Points as well)

Implementation Time

Now what is Bridge Tree?

Bridge Tree : If each bridge component of a given graph is shrinked into/represented as a single node, and these nodes are connected to each other by the bridge edges which separated these components, then the resulting tree formed is called a Bridge Tree.

Examples.

What are its properties?

1. Each edge in the normal graph G, is **either a bridge tree edge or part of one of the bridge components.**
2. The Bridge Tree is a Tree (Obvious from naming but should prove it anyways)
3. Number of bridges in a graph < **N**

Proof of (2) and (3)

Poll Question

1. Within a bridge component, if I pick any pair of nodes (u, v). Will there always be a simple cycle crossing both of them ? (True / False)

2. The bridge tree of a shape-8 graph will look like:

- Two nodes connected by an edge
- Same shape 8 graph
- A single node
- Same shape-8 graph without one of the edges

More properties

4. Within a bridge component, there is at least one way to **orient all the edges such that there is a simple path from any node to any node within the component. (Non-trivial)**
5. Within a bridge component, for any pair of nodes **(u, v)** there must be a **simple cycle between these two nodes. (Non-trivial)**

Proof of (4) and (5)

How do we make the bridge tree fast?

1. Run bridge finding algorithm to find all the bridges. $O(V + E)$
2. Remove all the bridges from G
3. In the resulting graph, the nodes in two different bridge components now look disjoint
4. So just label all the nodes with their component id.
5. Let the total number of these components be **K**
6. Now add back the bridges into a new graph with these **K** nodes and you get **B = (K, bridges)** as your bridge tree

Runtime: $O(V + E)$ or $O((V + E)\log E)$ depending on how you implement it.

Implementation Time

Easy Problems (1)

Q. Given an undirected connected graph with N nodes and M edges. You can add at-most 1 edge in the graph between any two nodes. **Find the minimum number of bridges in the resulting graph.**

Easy Problems (2)

Q. Given undirected G = (V, E), is there a pair of nodes (s, t), such that there are **>= 3 vertex-disjoint paths between s and t.**

Hard Problems (1)

Q. Given an undirected G = (V, E) and queries of the form Q = (u_i, v_i), **can we orient all the edges such that there is a path from u_i to v_i, for all i.** (Codeforces: Problem Link)

Hard Problems (2)

Q. Given an undirected G = (V, E) with cost associated to each edge. Find the best way to **remove a bridge edge and then add a new edge such that the graph still remains connected AND the sum of edge weights is maximized** (Codechef: Problem Link)

Further Readings

- Can read Tanuj's blog-post / watch his lecture video explaining this topic.
- Bridge tree was a way to compress the graph across "bridges"
- Block-Cut Tree is a way to compress the graph across "articulation points" (Can read up on this if interested)

Rule of Thumb: Block-Cut Tree is more powerful than Bridge Tree, but it is less intuitive and harder to code.

Thank You

Q&A