Graph Bridges & Articulation Points in DSA: Beginner to Advanced Guide

Subtitle: Learn how to find critical edges and vertices in graphs that affect connectivity.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Bridges and articulation points are **critical edges and vertices** in a graph. Removing them **increases the number of connected components**. Detecting them is essential in **network reliability, fault tolerance, and graph analysis**.

Step 1: Core Concepts

- **Bridge:** An edge whose removal increases the number of connected components
- **Articulation Point (Cut Vertex):** A vertex whose removal increases the number of connected components
- Use **DFS and low-link values** to identify bridges and articulation points
- **Time Complexity:** O(V + E)

Step 2: Graph Representation

- Use **adjacency list**

```
let graph = {

 0:[1,2],

 1:[0,2],

 2:[0,1,3],

 3:[2,4,5],

 4:[3,5],

 5:[3,4]

};
```

Step 3: DFS & Low-Link Values

- **Discovery Time (disc[u])**: time when node u is first visited
- **Low Value (low[u])**: earliest visited vertex reachable from u or its subtree

- **Bridge Condition:** low[v] > disc[u] for edge u-v
- **Articulation Point Condition:**
  - Root with **≥ 2 children**
  - Non-root: low[v] ≥ disc[u] for any child v

Step 4: Implementation in JavaScript

```
let time = 0;

let disc=[], low=[], parent=[], visited=[], bridges=[], aps=[];



function dfs(u){

 visited[u]=true;

 disc[u]=low[u]=++time;

 let children=0;



 for(let v of graph[u]){

  if(!visited[v]){

   parent[v]=u;

   children++;

   dfs(v);

   low[u]=Math.min(low[u], low[v]);



   // Bridge check

   if(low[v] > disc[u]) bridges.push([u,v]);



   // Articulation point check

   if(parent[u]===-1 && children>1) aps.push(u);

   if(parent[u]!==-1 && low[v]>=disc[u]) aps.push(u);

  }

  else if(v!==parent[u]){
```

```
      low[u]=Math.min(low[u], disc[v]);
    }
  }
}

// Initialize
let n = Object.keys(graph).length;
disc=Array(n).fill(0);
low=Array(n).fill(0);
parent=Array(n).fill(-1);
visited=Array(n).fill(false);

for(let i=0;i<n;i++) if(!visited[i]) dfs(i);

console.log("Bridges:", bridges);
console.log("Articulation Points:", [...new Set(aps)]);
```

Step 5: Applications

1. **Network reliability** – identify critical connections
2. **Graph analysis** – detect weak points in communication networks
3. **Competitive programming graph problems**
4. **Fault tolerance in distributed systems**

Step 6: Mini Exercises

1. Find **all bridges** in a sample graph
2. Find **all articulation points**
3. Solve **connectivity problems** after edge/vertex removal
4. Apply bridge/vertex detection to **real-world network scenarios**

Step 7: Common Mistakes

- Forgetting **to update low[u] on back edges**
- Misidentifying **root conditions** for articulation points
- Counting duplicate articulation points (use set)

- Miscalculating DFS discovery times

Step 8: Practice Plan for Bridges & Articulation Points

Day 1:

- DFS and low-link computation → 2–3 problems

Day 2:

- Bridges identification → 3–5 problems

Day 3:

- Articulation points and mixed graph queries → 3–5 problems

Key Takeaways

- Bridges and articulation points reveal **critical graph structures**
- DFS + low-link is essential for efficient detection
- Fundamental for **graph algorithms, network design, and contest problems**
- Practice helps develop **graph intuition and fault analysis skills**

Visit haas.dev for more guides, problem sets, and interview preparation resources on Bridges & Articulation Points.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app