**Bridges and Articulation Points**

**Iheb Gafsi**\*

INSAT Student

**Iheb.engineer@gmail.com**

**Definition:**

Bridges, also known as cut edges, are edges in an undirected graph that, if removed, increase the number of connected components in the graph. In other words, a bridge is an edge whose removal would disconnect the graph or split it into two or more separate components. Bridges are critical for maintaining connectivity in the graph, and their identification is essential for understanding the structure and robustness of the graph.

Articulation points, also known as cut vertices, are vertices in an undirected graph that, if removed along with their incident edges, increase the number of connected components in the graph. Similar to bridges, an articulation point is a vertex whose removal would disconnect the graph or split it into multiple components. Articulation points play a crucial role in determining the connectivity of the graph and identifying important nodes that act as "hinges" between different parts of the graph.

**Use cases:**

Bridges and articulation points have significant applications in graph analysis and network-related scenarios. Bridges are critical edges in a graph whose removal disconnects the graph, making them essential in network design, social network analysis, and transportation planning. On the other hand, articulation points are vertices whose removal disconnects the graph into multiple components. They are used to assess network reliability, optimize internet routing, and plan emergency responses. Identifying and understanding bridges and articulation points help in improving network performance and designing resilient and efficient systems.

**Algorithm For Finding Bridges:**

1. #Variables

2. graph = adjacency list

3. #Global variables needed for the Bridges Search Algorithm

4. id = 0

5.

6. # DFS Algorithm

7. def dfs(graph, i, p, visited, low\_links, ids, bridges):

8.     global id

9.     id+=1

10.     visited[i] = True

11.     low\_links[i] = ids[i] = id

12.     for arg in graph[i]:

13.         if arg == p: continue

14.         if not visited[arg]:

15.             dfs(graph, arg, i, visited, low\_links, ids, bridges)

16.             low\_links[i] = min(low\_links[i], low\_links[arg])

17.             if ids[i] < low\_links[arg]:

18.                 bridges.append((i, arg))

19.         else:

20.             low\_links[i] = min(low\_links[i], low\_links[arg])

21.

22.

23. # Bridges Search Algorithm

24. def bridges\_search(graph):

25.     n = len(graph)

26.     visited = [False] \* n

27.     low\_links = [0] \* n

28.     ids = [0] \* n

29.     bridges = []

30.     for i in range(n):

31.         if not visited[i]: dfs(graph, i, -1, visited, low\_links, ids, bridges)

32.     return bridges

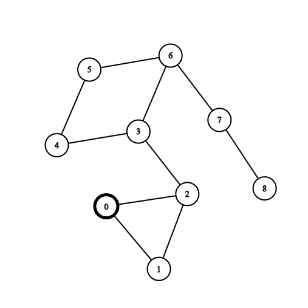
33.

34. print(bridges\_search(graph))

35.

**Example:**

Here’s a small example illustrating an example of input outputs for the Bridges Finding Algorithm:



We will use the Python code down below to outline the output of the algorithm on this graph:

1. #Variables

2. graph =[

3.     [1, 2],

4.     [0, 2],

5.     [0, 1, 3],

6.     [2, 4, 6],

7.     [3, 5],

8.     [4, 6],

9.     [3, 5, 7],

10.     [6, 8],

11.     [7]

12. ]

13. #Global variables needed for the Bridges Search Algorithm

14. id = 0

15. # DFS Algorithm

16. def dfs(graph, i, p, visited, low\_links, ids, bridges):

17.     global id

18.     id+=1

19.     visited[i] = True

20.     low\_links[i] = ids[i] = id

21.     for arg in graph[i]:

22.         if arg == p: continue

23.         if not visited[arg]:

24.             dfs(graph, arg, i, visited, low\_links, ids, bridges)

25.             low\_links[i] = min(low\_links[i], low\_links[arg])

26.             if ids[i] < low\_links[arg]:

27.                 bridges.append((i, arg))

28.         else:

29.             low\_links[i] = min(low\_links[i], low\_links[arg])

30. # Bridges Search Algorithm

31. def bridges\_search(graph):

32.     n = len(graph)

33.     visited = [False] \* n

34.     low\_links = [0] \* n

35.     ids = [0] \* n

36.     bridges = []

37.     for i in range(n):

38.         if not visited[i]: dfs(graph, i, -1, visited, low\_links, ids, bridges)

39.     return bridges

40.

41. print(bridges\_search(graph))

The corresponding output is:

Python >> [(7, 8), (6, 7), (2, 3)]

**Algorithm For Finding Articulation Points:**

The process of finding an articulation point is similar to the one of finding bridges however there are exceptions where the articulation point is trapped into a cycle or it’s a singleton so we will have to put some changes in the algorithm.

1. #Variables

2. graph =adjacency list

3. #Global variables needed for the Articulation Points Search Algorithm

4. id = 0

5. out = 0

6. # DFS Algorithm

7. def dfs(graph, r, i, p, visited, low\_links, ids, art):

8.     global id, out

9.     visited[i] = True

10.     low\_links[i] = ids[i] = id

11.     id += 1

12.     for arg in graph[i]:

13.         if arg == p: continue

14.         if not visited[arg]:

15.             dfs(graph, r, arg, i, visited, low\_links, ids, art)

16.             low\_links[i] = min(low\_links[i], low\_links[arg])

17.             if ids[i] <= low\_links[arg]:

18.                 art[i] = True

19.         else:

20.             low\_links[i] = min(low\_links[i], low\_links[arg])

21. # Articulation Points Search Algorithm

22. def art\_search(graph):

23.     global out

24.     n = len(graph)

25.     visited = [False] \* n

26.     low\_links = [0] \* n

27.     ids = [0] \* n

28.     art = [False] \* n

29.     for i in range(n):

30.         if not visited[i]:

31.             out = 0

32.             dfs(graph, i, i, -1, visited, low\_links, ids, art)

33.             art[i] = out > 1

34.     return art

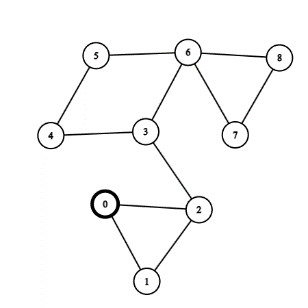
35.

36. print(art\_search(graph))

37.

**Example:**

Here’s a small example illustrating an example of input outputs for the Articulation points Finding Algorithm:



We will use the Python code down below to outline the output of the algorithm on this graph:

1. #Variables

2. graph =[

3.     [1, 2],

4.     [0, 2],

5.     [0, 1, 3],

6.     [2, 4, 6],

7.     [3, 5],

8.     [4, 6],

9.     [3, 5, 7, 8],

10.     [6, 8],

11.     [6, 7]

12. ]

13. #Global variables needed for the Articulation Points Search Algorithm

14. id = 0

15. out = 0

16. # DFS Algorithm

17. def dfs(graph, r, i, p, visited, low\_links, ids, art):

18.     global id, out

19.     visited[i] = True

20.     low\_links[i] = ids[i] = id

21.     id += 1

22.     for arg in graph[i]:

23.         if arg == p: continue

24.         if not visited[arg]:

25.             dfs(graph, r, arg, i, visited, low\_links, ids, art)

26.             low\_links[i] = min(low\_links[i], low\_links[arg])

27.             if ids[i] <= low\_links[arg]:

28.                 art[i] = True

29.         else:

30.             low\_links[i] = min(low\_links[i], low\_links[arg])

31. # Articulation Points Search Algorithm

32. def art\_search(graph):

33.     global out

34.     n = len(graph)

35.     visited = [False] \* n

36.     low\_links = [0] \* n

37.     ids = [0] \* n

38.     art = [False] \* n

39.     for i in range(n):

40.         if not visited[i]:

41.             out = 0

42.             dfs(graph, i, i, -1, visited, low\_links, ids, art)

43.             art[i] = out > 1

44.     return art

45. print(art\_search(graph))

The corresponding output is:

Python >> [True, False, True, True, False, False, False, False, False]

