CSC11F: Advanced Data Structures and Algorithms
Articulation Points and Bridges

Yutaka Watanobe (142-A, yutaka@u-aizu.ac.jp)

1 Introduction

By modeling a real world object in a graph structure, important properties of the target can be discovered by algorithms. Detecting vertices or edges which have a special property is one of typical problems.

In this lecture, we will study the way to find articulation points and bridges in a given graph G based on depth-first search algorithm.

2 Articulation Points

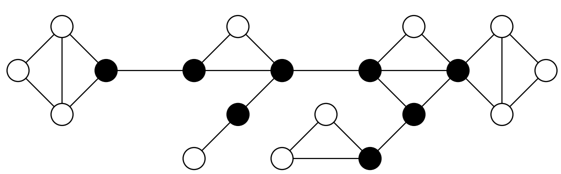

Figure 1: An example of articulation points

A vertex u is called an articulation point or disconnection point of a graph G if it is disconnected from the subgraph obtained by deleting the vertex u and all the edges coming from u in the connected graph G. For example, In Figure 1, black vertices are articulation points.

3 Bridges

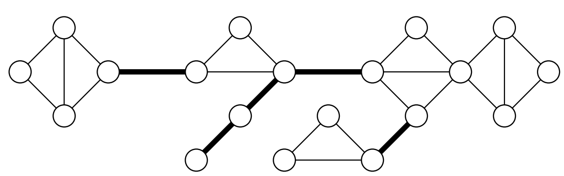

Figure 2: An example of bridges

On the other hand, an edge e is called a bridge of a graph G if it is disconnected from the subgraph obtained by deleting the edge e. For example, In Figure 2, bold edges are bridges.

4 Naive Algorithm

To find the articulation points in a given graph G, the following algorithm can be considered, which simply determines the connectivity of the graph from which it is removed, for each vertex.

```
getArticulationPoints():
    for each u in G:
        // remove vertex u and related edges from G
        G' = G - u
        Check connectivety of G' by DFS
        if G' is connected:
            u is NOT an articulation point
        else:
            u is an articulation point
```

Bridges can also be detected by the similar algorithm.

However, this is an $O(|V|^2)$ algorithm where $|V|$ is the number of vertices in G that performs depth-first search (DFS) or breadth-first search (BFS) every time, which is not efficient. The following DFS can be applied to efficiently detect all articulation points and bridges in a connected graph G.

5 Depth First Search

A single depth-first search can find values for the following variables:

- prenum[u]: DFS is performed starting from any vertex in G, and the order in which each vertex u is visited (discovered) is recorded in prenum[u].
- parent [u]: Record the parent of u in the tree T generated by DFS in parent [u]. Let T be the DFS Tree.
- lowest [u]: For each vertex u, compute the lowest [u] as the minimum of the following items:
    1. prenum[u].
    2. prenum[v] at vertex v if there exists a backedge(u, v) of G. A backedge (u, v) is an edge of G from vertex u to vertex v in T that does not belong to T.
    3. lowest [x] for all children x of vertex u belonging to T.

This algorithm is implemented by the following pseudo code.

```
dfs(u, p):
    prenum[u] = lowest[u] = t
    t++
    visited[u] = true

    for each v in G.adjList[u]:
        if !visited[v]:
            parent[v] = u
            dfs(v, u)
            lowest[u] = min(lowest[u], lowest[v])
        elif v != p:
            // edge (u, v) is a Back-edge
            lowest[u] = min(lowest[u], prenum[v])
```

Based on these variables, the articulation points are determined by the following conditions:

1. If the root r of T has two or more children (necessary and sufficient condition), then r is a articulation point.
2. For each vertex u, let p be the parent of u (parent $[u]$). If prenum $[p] \leq$ lowest $[u]$ (necessary and sufficient condition), then p is an articulation point (if p is the root, then we use the condition 1). This indicates that there is no edge from vertex u or the descendant of u at T to the ancestor of vertex p.

Articulation points are detected by the following algorithm.

```
art_points():
    t = 1
    dfs(0, -1) // root is 0

    art_points = empty
    nc = 0     // the number of chilren of the root
    for u = 1 to |V|-1:
        p = parent[u]
        if p == 0:
            nc++
        elif prenum[p] <= lowest[u]:
            art_points.insert(p)

    if nc > 1:
        // root is an articulation point
        art_points.insert(0)

    print contents of art_points
```

On the other hand, in the similar manner, bridges are detected by the following algorithm.

```
bridges():
    t = 1
    dfs(0, -1) // root is 0

    bridges = empty
    for u = 1 to |V|-1:
        if lowest[u] == prenum[u]:
            bridges.insert(edge(u, parent[u]))

    print contents of bridges
```

Let's see some concrete examples:

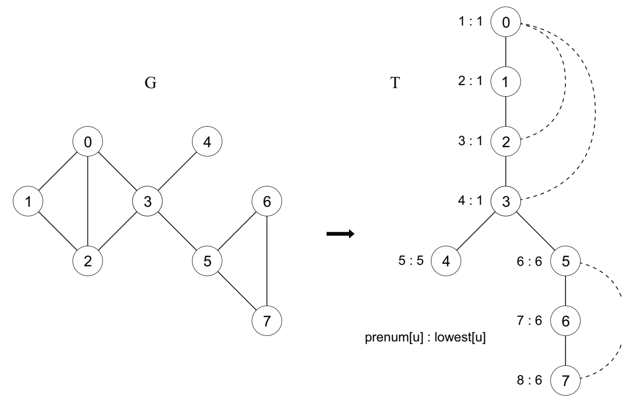

Figure 3: Articulation points detected by DFS (1)

Figure 3 shows a graph G and the DFS Tree T obtained by performing DFS from vertex 0 in G. Back edges at T are represented by dotted lines, with prenum$[u]$: lowest $[u]$ to the left of each vertex u, respectively.

prenum$[u]$ is the order in which each vertex u is visited in the DFS (preorder): $0 \to 1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 7$.

The lowest $[u]$ is "determined" in DFS by the order in which the visits of each vertex u are "completed" (postorder) : $4 \to 7 \to 6 \to 5 \to 3 \to 2 \to 1 \to 0$.

From the condition 1, vertex 0 (the root of T) is not an articulation point since the number of children of vertex 0 is one.

From the condition 2, we check whether prenum$[p] \leq$ lowest$[u]$ is satisfied where p is the parent of u.

Let's focus on vertex 5 (whose parent is vertex 3). Since prenum$[3] \leq$ lowest $[5](4 \leq 6)$, vertex 3 is an articulation point. This shows that there is no edge from vertex 5 or any descendant of vertex 5 tracing down any number of vertices in T to the ancestor of vertex 3.

Let's focus on vertex 2 (parent is vertex 1). Vertex 1 is not an articulation point because it does not satisfy prenum[1] $\leq$ lowest [2]. This indicates that there is an edge from vertex 2 or a descendant of vertex 2 to the ancestor of vertex 1.

Let's explore articulation points in practice using another example. Figure 4 shows another example of G and its DFT Tree T.

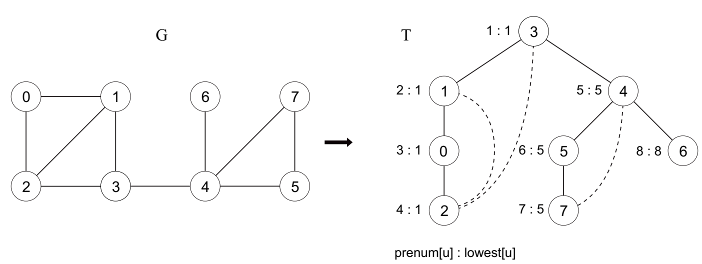

Figure 4: Articulation points detected by DFS (2)

6 Assignments

Place

https://onlinejudge.u-aizu.ac.jp/
beta/room.html#CSC11F_2023_Week_05

Duration

1 week

Problem A: Articulation Points

Find articulation points of a given undirected graph $G(V, E)$.

A vertex in an undirected graph is an articulation point (or cut vertex) iff removing it disconnects the graph.

Input

$|V|$ $|E|$
s_0 t_0
s_1 t_1
:
$s_{|E|-1|}$ $t_{|E|-1|}$

where $|V|$ is the number of vertices and $|E|$ is the number of edges in the graph. The graph vertices are named with the numbers $0, 1, ..., |V| - 1$ respectively.

s_i and t_i represent source and target verticess of i-th edge (undirected).

Output

A list of articulation points of the graph G ordered by name.

Constraints

- $1 \leq |V| \leq 100,000$
- $0 \leq |E| \leq 100,000$
- The graph is connected
- There are no parallel edges
- There are no self-loops

Sample Input 1

```
4 4
0 1
0 2
1 2
2 3
```

Sample Output 1

```
2
```

Sample Input 2

```
5 4
0 1
1 2
2 3
3 4
```

Sample Output 2

```
1
2
3
```

Problem B: Bridges

Find bridges of an undirected graph $G(V, E)$.

A bridge (also known as a cut-edge) is an edge whose deletion increase the number of connected components.

Input

$|V|$ $|E|$
s_0 t_0
s_1 t_1
:
$s_{|E|-1|}$ $t_{|E|-1|}$

where $|V|$ is the number of vertices and $|E|$ is the number of edges in the graph. The graph vertices are named with the numbers $0, 1, ..., |V| - 1$ respectively.

s_i and t_i represent source and target verticess of i-th edge (undirected).

Constraints

- $1 \leq |V| \leq 100,000$
- $0 \leq |E| \leq 100,000$
- The graph is connected
- There are no parallel edges
- There are no self-loops

Output

A list of bridges of the graph ordered by name. For each bridge, names of its end-ponints, source and target (source $<$ target), should be printed separated by a space. The sources should be printed ascending order, then the target should also be printed ascending order for the same source.

Sample Input 1

```
4 4
0 1
0 2
1 2
2 3
```

Sample Output 1

```
2 3
```

Sample Input 2

```
5 4
0 1
1 2
2 3
3 4
```

Sample Output 2

```
0 1
1 2
2 3
3 4
```