Lab08-Graph Exploration

CS214-Algorithm and Complexity, Xiaofeng Gao, Spring 2020.

∗ If there is any problem, please contact TA Yiming Liu.

∗ Name:Yijia Diao Student ID:518030910146 Email: diao_yijia@sjtu.edu.cn

1. **BFS Tree.** Similar to DFS, BFS yields a tree, (also possibly forest, but **just consider a tree** in this question) and we can define **tree, forward, back, cross** edges for BFS. Denote $Dist(u)$ as the distance between node u and the source node in the BFS tree. Please prove:

   (a) For both undirected and directed graphs, no forward edges exist in the graph.

   (b) There are no back edges in undirected graph, while in directed graph each back edge (u, v) yields $0 \leq Dist(v) \leq Dist(u)$.

   (c) For undirected graph, each cross edge (u, v) yields $|Dist(v) - Dist(u)| \leq 1$, while for directed graph, each cross edge (u, v) yields $Dist(v) \leq Dist(u) + 1$.

   Proof. (a) (By contradiction) Suppose there is a forward edge (u, v), then since u is connected with v, for undirected graph: $|Dist(v) - Dist(u)| = 1$, for directed graph: $Dist(v) - Dist(u) = 1$. According to the definition of forward edge, for directed and undirected graph, we have $|Dist(v) - Dist(u)| > 1$, contradiction.
   So we can conclude that for both undirected and directed graphs, no forward edges exist in the graph.

   (b) (By contradiction)
   1. For undirected graph: Suppose there is a back edge (u, v), then since u is connected with v, we have $Dist(u) - Dist(v) = 1$. According to the definition of back edge, $Dist(u) < Dist(v)$, contradiction.
   2. For directed graph: Suppose $Dist(v) > Dist(u)$, then v is a child or descendant of u. According to the definition of back edge, v is an ancestor of u, contradiction.
   Therefore we can conclude that the proposition is true.

   (c) (By contradiction)
   1. For undirected graph: Suppose that $|Dist(v) - Dist(u)| > 1$. According to the form of BFS tree, the edge (u, v) does not exist in the source graph, which leads to contradiction.
   2. For directed graph: Suppose $Dist(v) > Dist(u) + 1$. According to the form of BFS tree, the edge (u, v) does not exist in the source graph, which leads to contradiction.
   Therefore we can conclude that the proposition is true.

   □

2. **Articulation Points, Bridges, and Biconnected Components.** Let $G = (V, E)$ 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. Figure1 illustrates these definitions. We 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. Please prove:

   (a) The root of G_π is an articulation point of G if and only if it has at least two children in G_π.

   (b) An edge of G is a bridge if and only if it does not lie on any simple cycle of G.

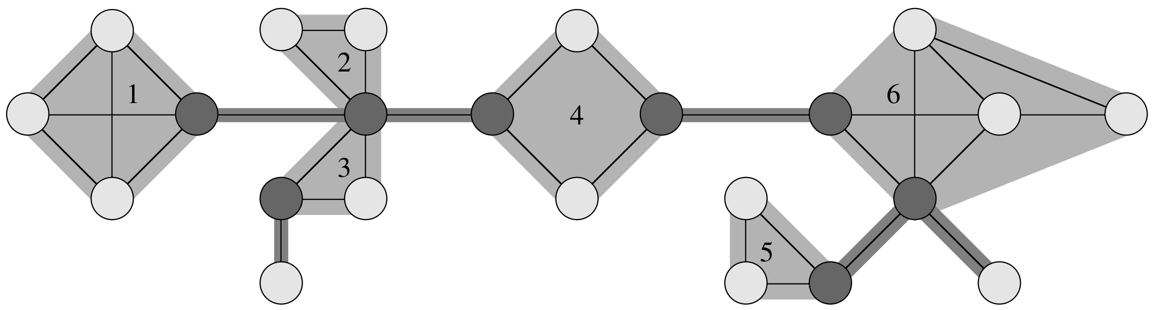

图 1: The definition of articulation points, bridges, and biconnected components. The articulation points are the heavily shaded vertices, the bridges are the heavily shaded edges, and the biconnected components are the edges in the shaded regions, with a *bcc* numbering shown.

(c) The biconnected components of G partition the nonbridge edges of G.

Proof. (a)

Lemma 1. *$\forall u, v \in$DFS tree, u and v are connected.*

We have proved the lemma in class.

"$\Rightarrow$" (By contradiction) Suppose the root of G_π has less than 2 children.

1. 0 child: the root doesn't connect to any other vertex. Contradiction with G is a connected graph.

2. 1 child: since with the removal of the root, its children is the new root of DFS tree, which indicates that the children is connected with all other nodes, according to Lemma 1. Therefore the graph is still connected. Contradiction with the definition of the articulation point.

"$\Leftarrow$" (By contradiction) Suppose the root is not an articulation point, which means if removes it G, the graph is still connected. Then remove it from G_π, then we have at least 2 new DFS tree, among whom are not connected, which indicates that the remaining nodes of G are not connected, according to Lemma 1. Contradiction.

Therefore, we can conclude that the proposition is true.

(b) "$\Rightarrow$" (By contradiction) Suppose the bridge edge (u, v) lies on a cycle of G. Then remove edge (u, v), u and v are still connected because of the cycle path. And we do not change other edges, so the connection of all other node pair does not changes, which means G is connected. Contradiction.

"$\Leftarrow$" (By contradiction) Suppose the edge (u, v) is not a bridge. Then remove edge (u, v), since u and v are not on any cycle, then u and v are not connected, contradiction with the definition of bridge.

Therefore, we can conclude that the proposition is true.

(c) 1. According to the proposition in (b) , the bridge edges do not lie in any biconnected component.

2. Now prove that the biconnected components is the partition of $G \backslash \{$bridge edges$\}$.

(**1**) Disjoint.

(By contradiction) Suppose that E_1, E_2 are two biconnected components, and $E_1 \neq E_2$, $E_1 \cap E_2 \neq \emptyset$. Then we have edge (u, v): $(u, v) \in E_1 \cap E_2$. Then (u, v) and any other edges $\in E_1 \cup E_2$ lie in a common simple cycle.

Assume that $E_1 \backslash E_2 \neq \emptyset$ (equals to $E_2 \backslash E_1$'s case). Then $\forall$ edge $(u_1, v_1) \in E_1 \backslash E_2$, $\forall$ edge $(u_2, v_2) \in E_2$, we can find cycle (a set of edges) $C_1 : (u_1, v_1), (u, v) \in C_1$,

$C_2 : (u_2, v_2), (u, v) \in C_2$. Then consider $C_1 \cup C_2 \backslash (u, v)$, which means the set of two different paths between u and v, thus $C_1 \cup C_2 \backslash (u, v)$ is also a cycle. So $(u_1, v_1), (u_2, v_2) \in E_2$, which leads to contradiction.
So we can conclude that $E_1 \cap E_2 = \emptyset$
(2) $\forall E$ is biconnected components: $\cup E = G \backslash \{\text{bridge edges}\}$.
(By contradiction) Suppose that there is an edge $(u, v) \in G$ that does not lie in any biconnected component, also isn't a bridge. Then according to the proposition 2b, (u, v) lies on a simple cycle of G, and we can find another edge (u', v') on the same cycle. Then $\exists$ cycle $C : (u, v), (u', v') \in C$, which means (u, v) lies on a biconnected component, contradiction.
So we can conclude: $\forall E$ is biconnected components: $\cup E = G \backslash \{\text{bridge edges}\}$

□

3. Suppose $G = (V, E)$ is a **Directed Acyclic Graph** (DAG) with positive weights $w(u, v)$ on each edge. Let s be a vertex of G with no incoming edges and assume that every other node is reachable from s through some path.

   (a) Give an $O(|V| + |E|)$-time algorithm to compute the shortest paths from s to all the other vertices in G. Note that this is faster than Dijkstra's algorithm in general.

   (b) Give an efficient algorithm to compute the longest paths from s to all the other vertices.

Solution. (a) The algorithm below uses the method framed by BFS, and its time complexity is $O(|V|+|E|)$. Since it does not use the operation of EXTRACT-MIN and DECREASE-KEY, it's faster than Dijkstra's algorithm.

Algorithm 1: Shortest Path for DAG

```
Input: $G = (V, E), w(u, v), s$.
Output: An array of the shortest distance $d[|V|]$ from $s$ to all the other
        vertices in $G$.
foreach $v \in V$ do  $d[v] \leftarrow \infty$;
$d[s] \leftarrow 0$;
push($queue$,$s$);
while $queue$ is not empty do
    $v \leftarrow$ pop($queue$);
    foreach edge $(v, u) \in E$ do
        $d[u] \leftarrow \min(d[u], w(v, u) + d[v])$;
        push($queue$,$u$);
return $d[|V|]$;
```

(b) The algorithm is similar to Alg.1, and just change the judgment function of distance value.

Algorithm 2: Longest Path for DAG

```
Input: $G = (V, E), w(u, v), s$.
Output: An array of the longest distance $d[|V|]$ from $s$ to all the other vertices
        in $G$.
foreach $v \in V$ do  $d[v] \leftarrow 0$;
push(*queue*,$s$);
while *queue is not empty* do
    $v \leftarrow$ pop(*queue*);
    foreach *edge* $(v, u) \in E$ do
        $d[u] \leftarrow \max(d[u], w(v, u) + d[v])$;
        push(*queue*,$u$);
return $d[|V|]$;
```

□

Remark: You need to include your .pdf and .tex files in your uploaded .rar or .zip file.