Building Blocks

A connected graph H is *2-connected* (or *biconnected*) if for every vertex v, the graph $H - v$ (with vertex set $V - v$ and edge set $E - \{e | e = wv\ for\ some\ w \in V - v\}$) is connected.

If a tree T contains a vertex v with two neighbours u and w then since T contains no cycle, there is no path of $T - v$ from u to w, so T is not 2-connected. So, a tree is 2-connected if and only if it is a vertex or an edge. Actually something stronger is true:

Observation 1: Every edge of a 2-connected graph H with more than three vertices is on a cycle.

Proof: Let vw be an edge of H. Since H is connected there is another edge of H which has one of its endpoints in $\{v, w\}$. WLOG we let uv be this edge. Since $H - v$ is connected it contains a path P between v and u. Now there is a cycle whose edges are the edges of P and uv, vw. QED.

The *blocks* of a graph G are the maximal 2-connected subgraphs within it. The blocks of a graph may share vertices. For example a graph with edge set $\{uv, vw\}$ has two blocks uv and vw which intersect in the vertex v. However, we have:

Observation 2: If H_1 and H_2 are both 2-connected and intersect in at least 2 vertices then $H_1 \cup H_2$ is 2-connected. ($V(H_1 \cup H_2) = V(H_1) \cup V(H_2)$,$E(H_1 \cup H_2) = E(H_1) \cup E(H_2)$).

Proof: For any vertex v of $H_1 \cup H_2$, there is a vertex w in $V(H_1) \cap V(H_2) - v$. Since $H_1 - v$ is connected and $H_2 - v$ is connected for every vertex x in $V(H_1 \cup H_2) - v$ there is a path P_x from x to w in $H_1 - H_2$. So, for any x and y in $V(H_1 \cup H_2)$, there is a path of $H_1 \cup H_2 - v$ from x to y through w. I.e. $H_1 \cup H_2 - v$ is connected. The same argument shows that $H_1 \cup H_2$ is connected. Thus. $H_1 \cup H_2$ is 2-connected. QED.

Corollary 3: Any two distinct blocks H_1 and H_2 intersect in at most one vertex.

Proof: Otherwise, $H_1 \cup H_2$ is biconnected, contradicting the maximality of H_1. QED.

In particular we see that the blocks of G are edge disjoint and yield a partition of its edge set.

This fact allows us to develop an algorithm to test if a connected graph G with n vertices and m edges is biconnected in $O(n^2 m)$ time.

As a subroutine, we use a brute force $O(nm)$ algorithm which tests if a connected graph H (with n vertices and m edges) is bicoonected by testing the connectivity of $H - v$ for each vertex v.

We apply this algorithm to G. If we determine that G is 2-connected then the only block of G is G itself. otherwise we find a vertex v such that $G - v$ is disconnected. Now, each block H of G intersects only one component of $G - v$ as otherwise $H - v$ is disconnected. For each component U of $G - x$, we compute the blocks of $U + v$ (the graph obtained from U by adding v to the vertex set and every edge of G from v to $V(U)$ to the edge set). We note that a graph is a block of G precisely if it is a block of $U + v$ for some such U. So, combining the solutions to the subproblems yields the solution to our original problem. It is easy to prove by induction that we consider at most $n - 1$ subproblems when treating a problem with $n \geq 2$ vertices. So, this algorithm takes $O(n^2 m)$ time to compute the blocks of G.

To obtain a faster algorithm,we need the following result.

Observation 4: A connected graph H is biconnected precisely if for every two distinct edges of H, there is a cycle containing both e and f.

Proof: If H is not biconnected then $H - v$ is disconnected for some v. Since, H is connected there is an edge from v to each component of $H - v$. Let uv and vw be edges of G to different components of $H - v$. There is no path of $H - v$ joining u and w, so there is no cycle containing both uv and vw. So if for every pair e, f of edges of H, there is a cycle containing both e and f then H is 2-connected.

Now, suppose for a contradiction that H is biconnected and e and f are distinct edges such that no cycle contains both e and f. Let E_{yes} be the set of edges which appear in a cycle with e and let $E_{no} = E - E_{yes}$. Then, $e \in E_{yes}$ and $f \in E_{no}$. Since H is connected, there is a path P containing both e and f. So, there must be two consecutive edges uv and vw of P such that $uv \in E_{yes}$ and $vw \in E_{no}$. Let C be a cycle of H containing e and uv. Then, w is not a vertex of C as otherwise, the edges of C can be partioned into a pair of paths between v and w. But now, vw along with the path in this pair which contains e is a cycle containing both vw and e, a contradiction. There is a path of $H - v$ between w and u since $H - v$ is connected. Let Q be a subpath of this path which has one endpoint w, the other in C, and no other vertex in C. Then, C can be partitioned into two paths between the endpoints of Q. But Q together with the path in this pair containg e forms a cycle, a contradiction. So, if H is biconnected then every two edges of H lie on a cycle. QED

Corollary 5: Distinct edges e and f of G are in the same block if and only if there is some cycle of G containing both e and f.

Proof: By Observation 4, if e and f are in the same block of G then there is a cycle of H (and hence of G containing them both. On the other hand, if e and f are in some cycle C of G then they are both in the block which is the maximal 2-connected subgraph containing (the 2-connected graph) C. QED.

This corollary implies that if e and f lie together on a cycle and f and g

lie together on a cycle then e and g lie together on a cycle. Can you prove this result directly?

Finding Blocks Using DFS Trees

We recall that if T is a DFS tree of a connected graph G then every edge e in $E(G) - E(T)$ links some vertex of G with one of its ancestors in T. Exploiting this fact and Corollary 5, we can obtain an $O(n+m)$ time algorithm for finding the blocks of a connected graph with n vertices $\{1, .., n\}$ and m edges $\{e_1, ..., e_m\}$.

So we consider a DFS tree rooted at 1, and corresponding vectors π and f. Thus, for $i \neq 1$, $\pi[i]$ is the parent of i in T and for all i, $f[i]$ is the time at which we finished examining i. Recall that if i is a descendant of j then $f[i] < f[j]$.

In partitioning $E(G)$ into the edge set of the blocks of G, the following observation is useful.

Observation 6: If $e = ij$ is an edge of E but not of T and $f[i] < f[j]$ then e is in the same block as the edge $i\pi[i]$.

Proof: We know j is an ancestor of i so the unique path P of T between j and i uses the edge $i\pi[i]$. Now $E(P) + e$ is the edge set of a cycle and the result follows by Corollary 5. QED.

Observation 7: For any edge $i\pi[i]$ of T, exactly one of the following holds:

(A) there is an edge from a descendant of i (this includes i) to an ancestor of $\pi[\pi[i]]$ (this includes $\pi[\pi[i]]$) and hence $i\pi[i]$ and $\pi[i]\pi[\pi[i]]$ are in the same block, or

(B) every edge of G from a descendant of i is either to another descendant of i or to $p[i]$ and hence for any tree edge $j\pi[j]$ in the block containing $i\pi[i]$, $f[j] < f[i]$.

Proof: If there is an edge from a descendant j of i to an ancestor k of $\pi[\pi[i]]$ then the unique path P of T between j and k, together with jk forms a cycle containing both $\pi[i]i$ and $\pi[i]\pi[\pi[i]]$ and so (A) holds.

If there is no edge from a descendant of i to an ancestor of $\pi[\pi[i]]$ then since every edge joins a vertex to one of its ancestors, we see that the the descendants of i form the vertex set of a component of $G - \pi[i]$. Thus, for any biconnected subgraph H of G, either H contains no descendant of i or H contains no vertex other than $\pi[i]$ which is not a descendant of i. In particular, the block H containing $\pi[i]i$ contains only edges at least one of whose endpoints is a descendant of i, so (B) holds. QED.

Our algorithm for finding the blocks of G, constructs two arrays. The first HO contains for every vertex i the largest value l such that there is an edge of

G from a descendant of i to an ancestor j of i with $f[j] = l$ or the value zero if $i = 1$.

We construct HO by first setting $HO[i] = f[\pi[i]]$ for every i between 2 and n and then performing the following:

Sort the vertices in increasing order of $f[i]$.
For each vertex i in turn (under this order) except 1 do:
  For each vertex j adjacent to i, if $f[j] > HO[i]$ set $HO[i] = f(j)$.
  If $\pi[i] \neq 1$
    If $HO[i] > HO[\pi[i]]$ then set $HO[\pi[i]] = HO[i]$.

It is clear that this dynamic programming algorithm performs as desired in $O(n+m)$ time.

We now consider the vertices of T other than the root in decreasing order of $f[i]$(thus we sort the vertices other than 1 in decreasing order of $f[i]$). We label them with positive integers so that the following holds. When we come to label an i if $\pi[i]i$ is in a block with an already labelled edge $\pi[j]j$ then we label i with the same label as j . Otherwise we label i with the lowest unused positve integer.

By Observation 7, we can do so by setting $B = 0$, and considering the vertices of T other than 1 in the given order, and for each vertex doing the following:

If $HO[i] = \pi[i]$ set $B = B + 1, Label[i] = B$. Otherwise set $Label[i] = Label[\pi[i]]$.

Now, we have partitioned the edges of T according to the blocks they are in, so to partition the edges of G we need only traverse the edge list, doing the following for each edge $e_i = jk$.

If $f[j] < f[k]$ then $Block[e_i] = Label[j]$ Otherwise $Block[e_i] = Label[k]$.

This yields a partition of the edges of G according to the blocks they are in. We only need to sort a set of n values between 1 and $2n$ and then do $O(n+m)$ work to update our arrays so the total time to carry out this algorithm is $O(n+m)$.