## 1 Vertex Connectivity

So far we've talked about connectivity for undirected graphs and weak and strong connectivity for directed graphs. For undirected graphs, we're going to now somewhat generalize the concept of connectedness in terms of network robustness. Essentially, given a graph, we may want to answer the question of how many vertices or edges must be removed in order to disconnect the graph; i.e., break it up into multiple components.

Formally, for a connected graph $G$, a set of vertices $S \subseteq G$ is a **separating set** if $G - S$ has more than one component. The set $S$ is also called a **vertex separator** or a **vertex cut**. The **connectivity** of $G$, $\kappa(G)$, is the minimum size of any $S \subseteq G$ such that $G - S$ is disconnected or has a single vertex; such an $S$ would be called a **minimum separator**. We say that $G$ is $k$-**connected** if $\kappa(G) \geq k$.

A hypercube $Q_k$ is $k$-connected.

The minimum number of edges in a $k$-connected graph on $n$ vertices is $\lfloor \frac{kn}{2} \rfloor$.

## 2 Edge Connectivity

We have similar concepts for edges. For a connected graph $G$, a set of edges $F \subseteq G$ is a **disconnecting set** if $G - F$ has more than one component. If $G - F$ has two components, $F$ is also called an **edge cut**. The **edge-connectivity** if $G$, $\kappa'(G)$, is the minimum size of any $F \subseteq G$ such that $F - S$ is disconnected or has a single vertex; such an $F$ would be called a **minimum cut**. A **bond** is a *minimal* non-empty edge cut; note that a bond is not necessarily a minimum cut. We say that $G$ is $k$-**edge-connected** if $\kappa'(G) \geq k$.

For a simple graph, $\kappa(G) \leq \kappa'(G) \leq \delta(G)$.

A hypercube $Q_k$ is also $k$-edge-connected.

To find a minimum cut, we can use a randomized algorithm that performs iterative and randomly selected edge contractions until only two vertices remain in the contracted graph. The edges between these two vertices represent a cut in the original graph. Iterating this procedure a sufficient number of times gives us a high probability that we have found a minimum cut. This is called **Karger's Algorithm**. The **Stoer-Wagner Algorithm** can be used to find an exact solution; however, we're going to omit its discussion for now.

```
procedure KARGERS(Graph $G(V, E)$)
    $M \leftarrow E(G)$                                   ▷ Minimum cut - just initialize as max size
    for Some number of iterations do
        $C \leftarrow G$                                  ▷ $C$ will be contracted graph
        while $|V(C)| > 2$ do
            $e \leftarrow$ selected randomly from $e \in E(C)$
            $C \leftarrow C \cdot e$                      ▷ Do contraction
        if $|E(C)| < M$ then
            $M \leftarrow E(C)$      ▷ New minimum cut is edges in $C$ representing a cut in $G$
    return $M$
```

## 3 Biconnectivity

A graph that has no cut vertices is also called **biconnected**. We differentiate between 2-connected here in that the graphs $K_1$ and $K_2$ would also be considered biconnected even if they aren't 2-connected. The **biconnected components** (BiCCs) of a connected (but not necessarily biconnected) graph are the maximal subgraphs of the graph that are themselves biconnected. These are also called **blocks**. A vertex that connects to different blocks is called an **articulation point** or a **cut-vertex**. A **block-cutpoint graph** is a bipartite graph where one partite set consists of cut-vertices and one partite set consists of contracted representations of of every BiCC.

### 3.1 Hopcroft-Tarjan BiCC Algorithm

An optimal algorithm for determining graph biconnectivity is the **Hopcroft-Tarjan** BiCC Algorithm. For now, we're going to only consider the simplest variant of the algorithm, wherein we output a vertex if we determine that it is an articulation point. This algorithm functions very similarly to Tarjan's SCC algorithm. We perform a DFS tracking for each vertex a *depth*, i.e. the order that vertices are first visited, and a *low point*, i.e. the lowest value depth that is reachable through following children (descendents in DFS tree) from a vertex.

The main idea is that if a vertex is able to reach another vertex with a lower depth by following child edges, then these vertices are both part of the same biconnected component. A non-root vertex is an articulation point if it has some child that has a low point greater than or equal to the depth of the vertex. A root vertex is an articulation point if it has multiple children; note that a root can have a children with a low point equal to the root's depth yet not be an articulation point.

```
procedure HopcroftTarjan(Graph $G(V, E)$)
    for all $v \in V(G)$ do        ▷ Assume all arrays and variables are globally accessible
        $depth(v) \leftarrow -1$
        $low(v) \leftarrow -1$
    $curDepth \leftarrow 1$        ▷ DFS order counter
    $artPoints \leftarrow \emptyset$        ▷ Articulation vertices
    for all $v \in V(G)$ do
        if $depth(v) = -1$ then
            BiCC-DFS($v$)
    return $artPoints$
```

```
procedure BiCC-DFS(Vertex $v$)
    $depth(v) \leftarrow curDepth, curDepth \leftarrow curDepth + 1$
    $low(v) \leftarrow depth(v)$
    $children \leftarrow 0, isArt \leftarrow$ false
    for all  do$u \in N(v)$
        if $depth(u) \neq -1$ then
            $parent(u) \leftarrow v$
            BiCC-DFS($u$)
            $children \leftarrow children + 1$
            if $low(u) \geq depth(v)$ then
                $isArt \leftarrow$ true
            $low(v) \leftarrow \min(low(v), low(u))$
        else if $u \neq parent(v)$ then
            $low(v) \leftarrow \min(low(v), depth(u))$
        if $parent(v) = \emptyset$ and $children > 1$ then
            $artPoints \leftarrow v$        ▷ Root with multiple children
        else if $parent(v) \neq \emptyset$ and $isArt =$ true then
            $artPoints \leftarrow v$
```