Section 2.2

Connectivity

IDEA OF DEPTH-FIRST SEARCH

- Select vertex v_0 and visit any vertex adjacent to v_0, say v_1.
- Next visit a vertex adjacent to v_1 that has not been visited.
- Continue until a vertex v_k is reached with the property that all of its neighbors have been visited.
- Backtrack to the last vertex visited prior to v_k, say v_{k-1} and visit any new vertices neighboring it. If none exist, backtrack until we find a vertex with unreached neighbors.
- When we backtrack to v_0 and find it has no unvisited neighbors, we have visited all possible vertices reachable from v_0.

PROPERTIES OF DFS

- The set of edges formed are the edges of a tree.
- If the graph still has vertices that are unvisited, we can choose one of the vertices and start the DFS again. (If this happens, the graph is disconnected.)
- When all vertices have been visited, the edges used in performing these visit are the edges of a forest.

DFS PARTITIONS EDGES

- The DFS algorithm partitions the edge set into two sets T (those edges contained in the forest) called **tree edges**. The remaining edges $B = E - T$ are called **back edges**.
- The set B can be partitioned further when applying DFS to a digraph:
  - B_1 is the set of back arcs that join two vertices y and x where $e = y \to x$ along some path from v_0 to y in the DFS tree that begins with v_0.
  - F is the set of forward arcs that join two vertices x and y where $e = x \to y$ along some path from v_0 to y in the DFS tree that begins with v_0.
  - C is the set of arcs in B that join two vertices joined by a unique tree path that contains v_0. The edges of C are called **cross edges**, since they are edges between vertices that are not descendants of one another in the DFS tree.

NUMBERING VERTICES IN DFS

While performing a DFS, we shall number the vertices v with an integer $n(v)$ which represents the order in which the vertices are first encountered during the search.

DEPTH-FIRST SEARCH ALGORITHM

Algorithm 2.2.1 Depth-First Search (DFS).

Input: A graph $G = (V, E)$ with distinguished vertex x.

Output: A set T of tree edges and an ordering $n(v)$ of the vertices.

Method: Use a label $m(e)$ to determine if an edge has been examined Use $p(v)$ to record the previous vertex to v in a search.

DFS (CONCLUDED)

1. For each $e \in E$, do the following: Set $m(e) \leftarrow$ "unused."
   Set $T \leftarrow \emptyset$, $i \leftarrow 0$.
   For every $v \in V$, do the following: Set $n(v) \leftarrow 0$.
2. Let $v \leftarrow x$.
3. Let $i \leftarrow i + 1$ and let $n(v) \leftarrow i$.
4. If v has no unused incident edges, then go to step 6.
5. Find an unused edge $e = uv$ and set $m(e) \leftarrow$ "used." Set $T \leftarrow T \cup \{e\}$.
   If $n(u) \neq 0$, then go to step 4;
   else $p(u) \leftarrow v, v \leftarrow u$ and go to step 3
6. If $n(v) = 1$, then halt; else $v \leftarrow p(v)$ and go to step 4.

RECURSIVE VERSION OF DFS

Algorithm 2.2.2 Recursive Version of Depth-First Search.

Input: A graph $G = (V, E)$ with starting vertex x.

Output: A set T of tree edges and an ordering $n(v)$ of the vertices.

1. Let $i \leftarrow 1$ and let $T \leftarrow \emptyset$. For all $v \in V$, do the following: Set $n(v) \leftarrow 0$.
2. While for some $u \in V$, $n(u) = 0$, do the following: DFS(u).
3. Output T.

PROCEDURE DFS

Procdure DFS($\boldsymbol{v}$)

1. Let $n(v) \leftarrow i$ and $i \leftarrow i + 1$.
2. For all $y \in N(v)$, do the following:
   if $n(y) = 0$, then $T \leftarrow T \cup \{e = yv\}$
   DFS(y)
   end DFS

CONNECTIVITY

- The **connectivity** of G, denoted by $k(G)$, is the minimum number of vertices whose removal disconnects G or reduces it to a single vertex K_1.
- The **edge connectivity** of G, denoted by $k_1(G)$, is the minimum number of edges whose removal disconnects G.
- The graph G is ***n*-connected** if $k(G) \geq n$ and is ***n*-edge connected** if $k_1(G) \geq n$.

SEPARATING SETS

- A set of vertices whose removal increases the number of components in a graph is called a **vertex separating set** (or **vertex cut set**). If the cut set consists of a single vertex, it is a called **cut vertex**.
- A set of edges whose removal increases the number of components in a graph is called a **edge separating set** (or **edge cut set**). If the cut set consists of a single edge, it is a called **cut edge** or a **bridge**.

BLOCKS

A **block** of a graph G is a maximal 2-connected subgraph; that is, a 2-connected subgraph H of G that is not a proper subgraph of any other 2-connected subgraph of G.

A THEOREM ON CUT VERTICES AND BRIDGES

Theorem 2.2.1: In a connected graph G:

1. A vertex v is a cut vertex if and only if there exists vertices u and w $(u, w \neq v)$ such that v is on every $u - w$ path of G.
2. A edge e is a bridge if and only if there exists vertices u and w such that e is on every $u - w$ path of G.

A RELATIONSHIP BETWEEN CONNECTIVITY AND EDGE CONNECTIVITY

Theorem 2.2.2: For any graph G,

$$k(G) \leq k_1(G) \leq \delta(G).$$

A CHARACTERIZATION OF BRIDGES

Theorem 2.2.3: In a graph G, the edge e is a bridge if and only if e lies on no cycle of G.

INTERNALLY DISJOINT PATHS

Two $u - v$ paths P_1 and P_2 are **internally disjoint** if

$$V(P_1) \cap V(P_2) = \{u, v\}.$$

A CHARACTERIZATION OF 2-CONNECTED GRAPHS

Theorem 2.2.4 (Whitney): A graph G of order $p \geq 3$ is 2-connected if and only if any two vertices of G lie on a common cycle.

MENGER'S THEOREM

Theorem 2.2.5 (Menger's Theorem): For nonadjacent vertices u and v in a graph G, the maximum number of internally disjoint $u - v$ paths equals the minimum number of vertices that separate u and v.

A GENERALIZATION OF WHITNEY'S THEOREM

Theorem 2.2.6: A graph is k-connected if and only if all distinct pairs of vertices are joined by at least k internally disjoint paths.

AN EDGE ANALOG TO MENGER'S THEOREM

Theorem 2.2.7: For any two vertices u and v of a graph G, the maximum number of edge disjoint paths joining u and v equals the minimum number of edges whose removal separates u and v.