Overview

- Graphs provide a concise representation of a range of problems
  - Map coloring — more generally, resource contention problems
  - Networks — communication, traffic, social, biological, ...

Definition and Representations

A graph \( G = (V, E) \), where \( V \) is a set of vertices, and \( E \) a set of edges. An edge \( e \) of the form \((v_1, v_2)\) is said to span vertices \( v_1 \) and \( v_2 \). The edges in a directed graph are directed.

A \( G' = (V', E') \) is called a subgraph of \( G \) if \( V' \subseteq V \) and \( E' \) includes every edge in \( E \) between vertices in \( V' \).

Adjacency matrix

A graph \( (V = \{v_1, \ldots, v_n\}, E) \) can be represented by an \( n \times n \) matrix \( a \), where \( a_{ij} = 1 \) iff \((v_i, v_j) \in E\).

Adjacency list

Each vertex \( v \) is associated with a linked list consisting of all vertices \( u \) such that \((v, u) \in E\).

Note that adjacency matrix uses \( O(n^2) \) storage, while adjacency list uses \( O(|V| + |E|) \) storage. Both can represent directed as well as undirected graphs.

Depth-First Search (DFS)

- A technique for traversing all vertices in the graph
- Very versatile, forms the linchpin of many graph algorithms

\[
\text{dfs}(V, E) \]

\[
\text{foreach } v \in V \text{ do } \text{visited}[v] = \text{false} \\
\text{foreach } v \in V \text{ do } \\
\quad \text{if not visited}[v] \text{ then explore}(V, E, v)
\]

\[
\text{explore}(V, E, v) \\
\text{visited}[v] = \text{true} \\
\text{previsit}(v) \quad \quad \quad \quad /*A placeholder for now*/ \\
\text{foreach } (v, u) \in E \text{ do } \\
\quad \text{if not visited}[u] \text{ then } \text{explore}(G, V, u) \\
\text{postvisit}(v) \quad \quad \quad \quad /*Another placeholder*/
\]

CSE 548: (Design and) Analysis of Algorithms

Graphs

R. Sekar
Graphs, Mazes and DFS

If a maze is represented as a graph, then DFS of the graph amounts to an exploration and mapping of the maze.

DFS and Connected Components

A connected component of a graph is a maximal subgraph where there is path between any two vertices in the subgraph, i.e., it is a maximal connected subgraph.

DFS Numbering

Associate post and pre numbers with each visited node by defining previsit and postvisit

previsit(v) = clock
postvisit(v) = clock

clock++

clock++

Property

For any two vertices u and v, the intervals [pre[u], post[u]] and [pre[v], post[v]] are either disjoint, or one is contained entirely within another.
**DFS of Directed Graph**

![Directed Graph Diagram]

**DFS and Edge Types**

![DFS Tree Diagram]

**Back edges** lead to an ancestor in the DFS tree.

**Forward edges** lead from a node to a child descendant in the DFS forest.

**Tree edges** are actually part of the DFS forest.

**No cross edges in undirected graphs!**

Back and forward edges merge.

**Biconnectivity in Undirected Graphs**

**Definition (Biconnected Components)**

- An **articulation point** in a graph is any vertex \( a \) that lies on every path from between vertices \( v, w \).
- A **biconnected** graph contains no articulation points.
- A **biconnected component** of a graph is a maximal subgraph that is biconnected.

**Illustration of Biconnected Components**

- Each biconnected component given a different color
- Articulation points have multiple colors
Biconnected Graphs

- Biconnected components are not disjoint.
- Articulation points are duplicated across them.

Articulation points ≡ single-points of failure
Graph is disconnected if any of them go down.
- Between any two \( u, v \in V \) in a biconnected graph there are two node-disjoint paths (and hence a cycle).

Finding articulation points during DFS

During DFS visit of vertex \( v \), we want to decide if it is an articulation point or not. Divide \( V \) into to disjoint sets:
- \( V_i \) includes nodes inside the DFS tree rooted at \( v \), and
- \( V_o = V - V_i \) of nodes outside the DFS tree rooted at \( v \).

Observation (1)
Suppose that every inside node \( v_i \) (i.e., \( v_i \in V_i - \{v\} \)) can reach some outside node \( v_o \) (i.e., \( v_o \in V_o \)) without going through \( v \). Then \( v \) is not an articulation point.

Finding articulation points during DFS (2)

Proof:
Paths within \( V_o \): By properties of DFS tree, any two vertices in \( V_o \) can reach each other without going through \( v \).

Paths between \( v_i \) and \( V_o \): Once \( v_i \) can reach any node in \( V_o \), it can reach all other nodes in \( V_o \) without having to go through \( v \).

Paths within \( V_i \): Consider nodes \( v_i \) and \( v'_i \) in \( V_i - \{v\} \). By our assumption there is a path from \( v_i \) to some \( v_o \in V_o \), and another path from \( v'_i \) to some \( v'_o \in V_o \), neither involving \( v \). Consider the path \( v_i \) to \( v_o \) to \( v'_o \) to \( v'_i \), which does not pass through \( v \).

Thus, for every pair of vertices, there is a path between them that avoids \( v \), and hence \( v \) is not an articulation point.

Finding articulation points during DFS (3)

Observation (2)
If Observation (1) does not hold then \( v \) is an articulation point.

This means that there exists some \( v_i \) that cannot reach an outside node \( v_o \) without going through \( v \). By definition, this means that \( v \) is an articulation point.
Finding articulation points during DFS (3)

We combine and slightly strengthen Observations (1), (2), while omitting the proof, as it is essentially the same as before.

**Observation (3)**

Let $Y$ denote the set of ancestors of $v$ and $X$ its immediate children in the DFS tree. Now, $v$ is not an articulation point iff each $x \in X$ has a path to some $y \in Y$ without going through $v$.

- By focusing on just the children and ancestors of $v$, this makes it easier to decide on articulation points during DFS.
- Note that any such path from $x$ to $y$ will follow zero or more tree edges down, followed by a back edge.

Finding articulation points during DFS (4)

Note than any such path from $x$ to $y$ will follow zero or more tree edges down, followed by a back edge.

- If this path bypasses $v$, this is going to occur due to a single back-edge that goes to an ancestor of $v$. So there is no need to consider paths with multiple back-edges.
- Note: Pre-number increases while following tree edges down, while it decreases when a back edge is followed.

**Key Idea:** During DFS, for each vertex $x$, maintain the highest ancestor that can be reached from $x$ by following tree edges down and then a back edge.

Finding articulation points during DFS (5)

**Key Idea:** During DFS, maintain the highest ancestor reachable from $x$ by following tree edges down and then a back edge.

- This info is maintained in the array $low$.
- $low[x] \leq low[x']$ for every child $x'$ of $x$ in DFS tree.
  - This captures the fact that we can follow the tree edge down from $x$ to $x'$, then go to whichever ancestor is reachable from $x'$.
  - Algorithmically, let $low[x] = \min(low[x], low[x'])$ when $\text{explore}(x')$ returns
- $low[x] \leq \text{pre}[x'']$ for $x''$ adjacent to $x$ but not a parent or child in the DFS tree.
  - Algorithmically, set $low[x] = \min(low[x], \text{pre}[x''])$
  - By properties of DFS, $x''$ is either a descendant or ancestor of $x$.
    - As we are taking $\min$, statement effective only if $x''$ is an ancestor.

Finding articulation points during DFS (6)

**Key Idea:** Maintain $low$ during DFS

- When visiting $x$, initialize $low[x]$ to $\text{pre}[x]$.
- When a DFS call on a child $x'$ of $x$ returns, check if $low[x'] \geq \text{pre}[x]$.
  - If so, the highest ancestor $x'$ can reach is not higher than $x$, i.e., $x'$ cannot go to ancestors of $x$ without going through $x$.
  - So, mark $x$ as articulation point
- If an adjacent vertex $x''$ is already visited when $x$ considers it, set $low[x] = \min(low[x], \text{pre}[x''])$ unless $x''$ is parent of $x$

All these can be done during a DFS, while increasing the cost of each step by only a constant amount — so, overall complexity is $O(|E| + |V|)$
Directed Acyclic Graphs (DAGs)

A directed graph that contains no cycles. Often used to represent (acyclic) dependencies, partial orders,...

Property (DAGs and DFS)
- A directed graph has a cycle iff its DFS reveals a back edge.
- In a dag, every edge leads to a vertex with lower post number.
- Every dag has at least one source and one sink.

Strongly Connected Components (SCC)

Analog of connected components for undirected graphs

Definition (SCC)
- Two vertices u and v in a directed graph are connected if there is a path from u to v and vice-versa.
- A directed graph is strongly connected if any pair of vertices in the graph are connected.
- A subgraph of a directed graph is said to be an SCC if it is a maximal subgraph that is strongly connected.

SCCs are also similar to biconnected components!

SCC Example

The textbook describes an algorithm for computing SCC in linear-time using DFS.

Breadth-first Search (BFS)

- Traverse the graph by “levels”
  - BFS(v) visits v first
  - Then it visits all immediate children of v
  - then it visits children of children of v, and so on.
- As compared to DFS, BFS uses a queue (rather than a stack) to remember vertices that still need to be explored
BFS Algorithm

\textbf{bfs}(V, E, s)

\hspace{0.5em} \textbf{foreach} \ u \in V \ \textbf{do} \ \text{visited}[u] = false

\hspace{1.5em} q = \{s\}; \ \text{visited}[s] = true

\hspace{0.5em} \textbf{while} \ q \ \text{is nonempty} \ \textbf{do}

\hspace{1.5em} u = \text{dequeue}(q)

\hspace{1.5em} \textbf{foreach} \ \text{edge} \ (u, v) \in E \ \textbf{do}

\hspace{2.5em} \textbf{if not} \ \text{visited}[v] \ \textbf{then}

\hspace{3.5em} \text{queue}(q, v); \ \text{visited}[v] = true

Shortest Paths and BFS

BFS automatically computes shortest paths!

\textbf{bfs}(V, E, s)

\hspace{0.5em} \textbf{foreach} \ u \in V \ \textbf{do} \ \text{dist}[u] = \infty

\hspace{1.5em} q = \{s\}; \ \text{dist}[s] = 0

\hspace{0.5em} \textbf{while} \ q \ \text{is nonempty} \ \textbf{do}

\hspace{1.5em} u = \text{dequeue}(q)

\hspace{1.5em} \textbf{foreach} \ \text{edge} \ (u, v) \in E \ \textbf{do}

\hspace{2.5em} \textbf{if dist}[v] = \infty \ \textbf{then}

\hspace{3.5em} \text{queue}(q, v); \ \text{dist}[v] = \text{dist}[u] + 1

But not all paths are created equal! We would like to compute shortest weighted path — a topic of future lecture.

Graph paths and Boolean Matrices

A graph and its boolean matrix representation

\[ A = \begin{bmatrix}
0 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0
\end{bmatrix} \]
Graph paths and Boolean Matrices

- Let $A$ be the adjacency matrix for a graph $G$, and $B = A \times A$. Now, $B_{ij} = 1$ iff there is a path in the graph of length 2 from $v_i$ to $v_j$.
- Let $C = A + B$. Then $C_{ij} = 1$ iff there is a path of length $\leq 2$ between $v_i$ and $v_j$.
- Define $A^* = A^0 + A^1 + A^2 + \cdots$. If $D = A^*$ then $D_{ij} = 1$ iff $v_j$ is reachable from $v_i$.

\[
A = \begin{bmatrix}
0 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
A^2 = \begin{bmatrix}
1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
A^3 = \begin{bmatrix}
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]

Shortest paths and Matrix Operations

- Redefine operations on matrix elements so that $+$ becomes $\min$, and $\ast$ becomes integer addition.
- $D = A^*$ then $D_{ij} = k$ iff the shortest path from $v_j$ to $v_i$ is of length $k$. 