Chapter 23: Elementary Graph Algorithms

Adjacency-List Representation

1: [2, 6]
2: [3, 5]
3: []
4: [1, 3]
5: [4, 6]
6: [2]

The size is $\Theta(E + V)$

Suitable for \textit{sparse graphs}

\textbf{Adjacency-Matrix Representation}

the matrix $(a_{ij})$ such that $a_{ij} = 1$ if and only if there is an edge from $i$ to $j$ (between $i$ and $j$ for undirected graphs)

\[
\begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

The size is $\Theta(V^2)$

Suitable for \textit{dense graphs}

\textbf{Traversing all the nodes}

\textbf{Breadth-First Search:} the closer the earlier

\textbf{Depth-First Search:} follow a path as long as unvisited nodes are encountered.

\textbf{BFS + the minimum distance}

Start from $s$ and compute for each node $v$ its minimum distance $d(v)$ from $s$

\textbf{Idea:} For $i = 0, 1, \ldots$, use the nodes with distance $i$ to find the nodes with distance $i+1$.

Set a queue $Q$ to $\{s\}$, $d(v)$ to $+\infty$ for all $v$, and $d(s) = 0$

While $Q \neq \emptyset$, do:

- Pop an element $u$ from $Q$
- For each $v$ such that $(u, v) \in E$, if $d(v) = +\infty$ set $d(v) = d(u) + 1$ and push $v$ in $Q$
Correctness of the $d$-values
Let $\delta(v)$ be the shortest path length from $s$ to $v$.

**Theorem A** For each vertex $v$, the final value of $d[v]$ is $\delta(v)$.

**Proof Sketch** By induction on $t = d[v]$. The base case, $t = 0$, is trivial.

For the induction step, let $d[v] = t > 0$ and assume that the claim holds for all $t' < t$. Let $v$ be a node with $\delta(v) = t$. There is a node $u$ with $\delta(u) = t - 1$ such that $(u, v) \in E$. Applying the induction hypothesis to $u$ yields $d[v] \leq t$. Applying that to $v$ yields $d[v] \geq t$; otherwise, $\delta[v] \leq t - 1$. So, $d[v] = \delta(v)$.

**DFS**
Use recursive calls to a subroutine $\text{Visit}$

**time** $\defined$ the number of calls from $\text{Visit}$ or returns to $\text{Visit}$ that have been made so far

For each node $u$, compute:

- $d[u] \defined$ the time when $v$ is first visited
- $f[u] \defined$ the time when all its adjacent nodes have been examined
The main-loop:

- For all $u$, set $d[u] = \infty$, $\pi[u] = \text{NIL}$, and $time = 0$
- For each $u$, if $d[u] = \infty$ then call \textsc{Visit}(u)

\textsc{Visit}(u):

1. Increment $time$ and set $d[u] = time$
2. For each $v \in \text{Adj}[u]$, if $d[v] = \infty$ then set $\pi[v] = u$ and call \textsc{Visit}(v)
3. Increment $time$ and set $f[u] = time$

\textbf{Running Time Analysis}

- A call of \textsc{Visit} with respect to a node is exactly once and
- Each edge is examined exactly twice

\textbf{Q2. So what’s the running time?}

Again we can use a field $\pi$ to compute the predecessor, which induces a \textit{DFS tree}

<table>
<thead>
<tr>
<th>node</th>
<th>$\pi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u$</td>
<td>$v,x$</td>
</tr>
<tr>
<td>$v,x$</td>
<td>$u$</td>
</tr>
<tr>
<td>$w$</td>
<td>$z$</td>
</tr>
<tr>
<td>$y$</td>
<td>$v$</td>
</tr>
<tr>
<td>$z$</td>
<td>$y$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>node</th>
<th>$\pi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u$</td>
<td>$2/9$</td>
</tr>
<tr>
<td>$v,x$</td>
<td>$5/6$</td>
</tr>
<tr>
<td>$w$</td>
<td>$z$</td>
</tr>
<tr>
<td>$y$</td>
<td>$v$</td>
</tr>
<tr>
<td>$z$</td>
<td>$y$</td>
</tr>
</tbody>
</table>
**Parenthesis structure**

Let $I[u] = (d[u], f[u])$ for each $u$. Then for every $u, v$, $I[u] \cap I[v] = \emptyset$, $I[u] \subseteq I[v]$, or $I[v] \subseteq I[u]$.

**Classification of edges**

1. **Tree edges**: those on the tree
2. **Back edges**: non-tree edges connecting descendants to ancestors (self-loops, too)
3. **Forward edges**: non-tree edges connecting ancestors to descendants
4. **Cross edges**: the remainders

In DFS, when $e = (u, v)$ is first explored:
- $d[v] = \infty \Rightarrow e$ is a tree edge,
- $d[v] < f[v] = \infty \Rightarrow e$ is a back edge, and
- $f[v] < \infty \Rightarrow e$ is a forward or cross edge.

**Topological sort**

A linear ordering of the nodes in a DAG (directed acyclic graph) $G$ is a topological sort if for every $u$ and $v$ with $(u, v)$ in $G$, $u$ precedes $v$ in the order.

**TOPOLOGICAL-SORT($G$)**

1. call DFS($G$) to compute finishing times $f[v]$ for each vertex $v$
2. as each vertex $v$ is finished, insert $v$ onto the front of a linked list
3. return the linked list

**Q3.** What is a topological sort of these nodes?

**Q4.** What’s the running time?
**Strongly Connected Components**

\( u \sim v \overset{\text{def}}{=} \text{there is a path from } u \text{ to } v \text{ in } G \)

Two vertices \( u \) and \( v \) of a directed graph \( G \) are **strongly connected** if \( u \sim v \) and \( v \sim u \).

A strongly connected component of \( G \) is a maximal set \( S \) of vertices in \( G \) in which every two nodes are strongly connected.

A trivial algorithm: compute for each \( u \), \( W_u \overset{\text{def}}{=} \{ v \mid u \sim v \} \), and check for each \( u, v \), whether \( u \in W_v \) and \( v \in W_u \).

**Q5. What's the running time?**

* An \( O(E + V) \)-time method:

\( G^T \overset{\text{def}}{=} G \text{ with every edge reversed} \)

1. Call \( \text{DFS}(G) \) to compute \( f[u] \) for all \( u \).
2. Compute \( H = G^T \) so that the \( f \)-values are decreasing.
3. Call \( \text{DFS}(H) \).
4. Output the vertices of each DFS-tree of \( H \) as a separate strongly connected component.

**The \( f \)-values:**

\[
\begin{array}{ccccccc}
\end{array}
\]

**The DFS-trees of \( H \):**

\[
\begin{array}{ccccccc}
\end{array}
\]