10 Graph Algorithms

(These notes follow the development in Cormen, Leiserson, and Rivest.)

10.1 Definitions

• graph, directed graph (digraph), nodes, edges, subgraph

• connected component, strongly connected component

• directed acyclic digraph (DAG)

• path, cycle, clique, complete graph

• spanning tree, minimal spanning tree

• depth-first search, breadth-first search

• adjacency matrix
  – compressed format for dense graphs
  – requires search on rows or columns to find edges

• adjacency list
  – exactly as many storage nodes as edges
  – no search for edges and adjacent nodes
  – no “random” access for edge verification
10.2 Graph Problems

- explore all nodes (breadth first and depth first)
- find the shortest path from a given source to a given sink
- find the shortest path from a given source to all other nodes
- find all shortest paths from a given source to a given sink
- find all shortest paths from a given source to all other nodes
- find a shortest path between all pairs of nodes
- find an Eulerian circuit that uses each edge exactly once
- find a Hamiltonian circuit that uses each vertex exactly once
- any of the above problems with non-unit weights on the edges
- construction of graphs/trees to match pairwise weights between nodes

10.3 Breadth-First Search

Start with a source vertex

Keep three colors of vertices:

All vertices start out white

Color changes to grey when the vertices are discovered

Color changes to black when all adjacent vertices have been discovered

Use an adjacency list for representation
Breadth-First search in graph G from source vertex s
1    for each vertex u in V(G)\{s}\n2       {\n3         color(u) = white\n4         distance(u) = infinity\n5         predecessor(u) = NULL\n6       }\n7    color(s) = grey\n8    distance(s) = 0\n9    predecessor(s) = NULL\n10   queue = \{s\}\n11   while queue not empty\n12       {\n13         u = head(queue)\n14         for each v in adjacency(u)\n15            {\n16               if(color(v) == white)\n17                  {\n18                     color(v) = grey\n19                     distance(v) = distance(u) + 1\n20                     predecessor(v) = u\n21                     queue = queue + \{v\} /* i.e. enqueue */\n22                  }\n23            }\n24         color(u) = black\n25         queue = tailof(queue) /* i.e. dequeue */\n26       }\n
Analysis:
lines 1-6 take \( \Theta(|V|) \) time
loop 11-26 executes \( \Theta(|E|) \) time
queueing and unqueueing are constant time
10.4 Shortest Path Distances

Definition 10.1. A shortest-path distance $\delta(u, v)$ from a node $u$ to a node $v$ is the minimum number of edges necessary in any path from $u$ to $v$, or $\infty$ if no path from $u$ to $v$ exists.

Lemma 10.2. Let $G = (V, E)$ be any directed or undirected graph, and let $s \in V$ be any vertex. Then for any edge $(u, v) \in E$,

$$\delta(s, v) \leq \delta(s, u) + 1$$

Proof. If $u$ can be reached from $s$ then so can $v$. The shortest path from $s$ to $v$ can be no longer than the path from $s$ to $u$ followed by the edge $(u, v)$.

If $u$ cannot be reached from $s$, then $\delta(s, u) = \infty$ and the lemma is trivially true. \qed

Example: Consider, for a directed graph, the pathological case when the inequality is very much not sharp.

Lemma 10.3. Run BFS on a directed or undirected graph $G = (V, E)$ from a source $s$. Then for any node $u \in V$ we have

$$\text{distance}(u) \geq \delta(s, v)$$

Proof. Induct on nodes when they are first queued. At the initiation of BFS, we have

$$\text{distance}(s) = 0 = \delta(s, s)$$
and
\[ \text{distance}(v) = \infty \geq \delta(s, v) \]
for all other nodes \( v \).

Now when we discover a white vertex \( v \) when searching from node \( u \), we have
\[ \text{distance}(u) \geq \delta(s, u) \]
by the induction hypothesis, and then
\[ \text{distance}(v) = \text{distance}(u) + 1 \geq \delta(s, u) + 1 \geq \delta(s, v). \]

Since we only queue nodes when they are white, and change the color to grey when they are queued, we never change the distance value. \( \square \)

**Lemma 10.4.** Suppose that during BFS, the queue \( Q \) contains the vertices \((v_1, v_2, \ldots, v_n)\). Then
\[ \text{distance}(v_n) \leq \text{distance}(v_1) + 1 \]
and
\[ \text{distance}(v_i) \leq \text{distance}(v_{i+1}) \]
for all \( i < n - 1 \).

**Proof.** Induct on the number of queueing actions. Certainly when the queue holds only \( s \), the conclusion holds.

If the head \( v_1 \) of the queue is taken off, we have
\[ \text{distance}(v_n) \leq \text{distance}(v_1) + 1 \leq \text{distance}(v_2) + 1 \]
and everything stays ok.

When we put vertex \( v = v_{n+1} \) onto the queue, we are at that point looking at the adjacency list for vertex \( v_1 \). So at that point we have

\[
distance(v_{n+1}) = distance(v_1) + 1
\]

and we have

\[
distance(v_n) \leq distance(v_1) + 1 = distance(v_{n+1})
\]

so all the inequalities still hold.

\[\Box\]

**Theorem 10.5.** Given a directed or undirected graph on which we run BFS from a source \( s \). Then BFS discovers every vertex \( v \) that is reachable from the source and terminates with \( distance(v) = \delta(s, v) \) for all other vertices. Further, for any nonsource vertex \( v \), one of the shortest paths from \( s \) to \( v \) is the shortest path from \( s \) to \( \text{predecessor}(v) \) followed by the edge from \( \text{predecessor}(v) \) to \( v \).

**Proof.** Assume that a vertex \( v \) is unreachable from the start vertex \( s \). From Lemma 10.2 we have that \( distance(v) \geq \delta(s, v) = \infty \), so the BFS computation cannot set a finite distance value. Therefore only vertices with finite distance values are discovered by the computation, and thus only reachable vertices are discovered.
Now let $V_k$ be the set of vertices reachable at distance exactly $k$, thus

$$V_k = \{v \in V : \delta(s, v) = k\}.$$

We induct on $k$. Our inductive hypothesis is that at exactly one point during BFS we have

- that $v$ changes color to grey
- that $\text{distance}(v)$ is set to $k$
- that for $v \neq s$, $\text{predecessor}(v)$ is set to $u$ for some $u \in V_{k-1}$
- $v$ is inserted into the queue

The inductive hypothesis holds for $k = 0$ when $V_0 = \{s\}$. We now note that the queue is never empty until BFS terminates and that once $\text{distance}(u)$ and $\text{predecessor}(u)$ are set they are never changed. By Lemma ?? we have that the distances are nondecreasing.

Now consider an arbitrary $v \in V_k$. Then by Lemmas ?? and ?? we have that $v$ is not discovered (if it is ever discovered) until all the vertices of $V_{k-1}$ are on the queue.

Since $\delta(s, v) = k$ there must be a path from $s$ to $k$ of length $k$ edges, and thus there must be a vertex $u \in V_{k-1}$ and an edge $(u, v)$. WLOG we may assume that $u$ is the first such vertex whose color is changed to grey, so $u$ must be on the queue and thus must at some point appear as the head of the queue and be scanned by BFS. When this happens, $v$ is discovered. ($v$ could not have been discovered earlier since that would imply that $\delta(s, v) < k$.)
When \( v \) is discovered, \( \text{distance}(v) \) is set to \( k \), \( \text{predecessor}(v) \) to \( u \), and \( v \) is put on the queue.

This proves the inductive step.

Finally, we observe that if \( v \in V_k \) we have \( \text{predecessor}(v) \in V_{k-1} \). This means that a shortest path from \( s \) to \( v \) can be made by using the shortest path from \( s \) to \( \text{predecessor}(v) \) and then following the edge \( (\text{predecessor}(v), v) \). \( \square \)

One of the things that the BFS algorithm does is to construct the **predecessor subgraph** \( G_P = (V_P, E_P) \) such that

\[
V_P = \{s\} \cup \{v \in V : \text{predecessor}(v) \neq \text{NULL}\}
\]

and

\[
E_P = \{(\text{predecessor}(v), v) \in E : v \in V_P \setminus \{s\}\}
\]

The predecessor subgraph is a **breadth-first tree** if \( V_P \) contains all the vertices reachable from \( s \) and there is a unique simple path from \( s \) to \( v \) in \( G_P \) that is also a shortest path from \( s \) to \( v \) in \( G \).

**Lemma 10.6.** The BFS procedure constructs a predecessor subgraph that is a breadth-first tree.

**Proof.** BFS sets \( \text{predecessor}(v) \) to \( u \) only if \( (u, v) \) is an edge and if \( \delta(s, v) < \infty \), that is, if \( v \) is reachable from \( s \) and \( V_P \) consists of the vertices reachable from \( s \). Since \( G_P \) is a tree, there is a unique path from \( s \) to any \( v \) in the tree. We apply the previous theorem inductively to conclude that every such path is in fact a shortest path. \( \square \)
10.5 Depth-First Search

We keep a timestamp \texttt{firsttime} of the time when the color changes from white to grey and a timestamp \texttt{lasttime} of the time when the color changes from grey to black.

\begin{verbatim}
Depth-First Search
1 for each vertex u in V(G)
2 {
3    color(u) = white
4    predecessor(u) = NULL
5 }
6 time = 0
7 for each vertex u in V(G)
8 {
9    if(color(u) == white)
10       DFS_VISIT(u)
11 }

DFS-VISIT
1 color(u) = grey
2 firsthandtime(u) = time
3 time = time + 1
4 for each v in adjacency(u)
5 {
6    if(color(v) == white)
7       {  
8         predecessor(v) = u
9         DFS-VISIT(v)
10       }
11 }
12 color(u) = black
13 lasttime(u) = time
14 time = time + 1
14 return
\end{verbatim}
Analysis:

Once again, we execute the recursive loop at most once for each edge, and in fact we do execute once for each edge, so the running time is $\Theta(|V| + |E|)$.

**Theorem 10.7. (Nested Parenthesis Theorem)** In any DFS of a directed or undirected graph and any pair of vertices $u$ and $v$, then exactly one of the following holds:

1. the intervals $(\text{firsttime}(u), \text{lasttime}(u))$ and $(\text{firsttime}(v), \text{lasttime}(v))$ are entirely disjoint;

2. the interval $(\text{firsttime}(u), \text{lasttime}(u))$ is entirely contained within the $(\text{firsttime}(v), \text{lasttime}(v))$ and $u$ is a direct descendant of $v$ in the DF tree;

3. the interval $(\text{firsttime}(v), \text{lasttime}(v))$ is entirely contained within the $(\text{firsttime}(u), \text{lasttime}(u))$ and $v$ is a direct descendant of $u$ in the DF tree.

**Proof.** Assume that $\text{firsttime}(u) < \text{firsttime}(v)$ and that $\text{firsttime}(v) < \text{lasttime}(u)$. Then $v$ was discovered while $u$ was still grey, and hence $v$ is a descendant of $u$. Further, all the outgoing edges from $v$ have already been explored prior to $\text{lasttime}(u)$, processing from $v$ has been completed, and $v$ has been colored black, prior to $\text{lasttime}(u)$. That is, we have that $\text{lasttime}(v) < \text{lasttime}(u)$ and we are in case (3) above.

If, on the other hand, we have that $\text{firsttime}(u) < \text{firsttime}(v)$ and $\text{lasttime}(u) < \text{firsttime}(v)$, then we are in case (1).
Case (2) is the symmetric case to (3) under the conditions $\text{firsttime}(v) < \text{firsttime}(u)$ and that $\text{firsttime}(u) < \text{lasttime}(v)$. □

**Theorem 10.8.** Vertex $v$ is a proper descendant of vertex $u$ in the DF forest for a graph if and only if

$$\text{firsttime}(u) < \text{firsttime}(v) < \text{lasttime}(v) < \text{lasttime}(u)$$

**Theorem 10.9.** In a DF forest of a directed or undirected graph $G$, vertex $v$ is a descendant of vertex $u$ if and only if at time $\text{firsttime}(u)$ when vertex $u$ is discovered, vertex $v$ can be reached from $u$ along a path consisting only of white vertices.

**Proof.** $\Rightarrow$: Assume that $v$ is a descendant of $u$. Let $w$ be any vertex on the path from $u$ to $v$. Then $\text{firsttime}(u) < \text{firsttime}(w)$ by the previous theorem, so $w$ is white at time $\text{firsttime}(u)$.

$\Leftarrow$: Assume that a white path exists from $u$ to a vertex $v$, but in the DF forest it happens that $v$ is not a descendant of $u$. WLOG let $v$ be the first such vertex in the path to $v$. Let $w$ be the predecessor of $v$. Then by the previous theorem we have $\text{lasttime}(w) \leq \text{lasttime}(u)$. Further, $v$ must be discovered before $w$ is finished but after $u$ is discovered. Thus $\text{firsttime}(u) < \text{firsttime}(v) < \text{lasttime}(w) \leq \text{lasttime}(u)$ and hence the interval $(\text{firsttime}(v), \text{lasttime}(v))$ is entirely contained within the interval $(\text{firsttime}(u), \text{lasttime}(u))$. But then $v$ is in fact a descendant of $u$. □

**Definition 10.10.** A **tree edge** is an edge in the DF forest $G_F$. An edge $(u,v)$ is a tree edge if $v$ was first discovered by exploring edge $(u,v)$. 

Definition 10.11. A back edge is an edge \((u, v)\) connecting a vertex \(u\) to an ancestor \(v\).

Definition 10.12. A forward edge is a nontree edge \((u, v)\) connecting a vertex \(u\) to a descendant \(v\) in a DF tree.

Definition 10.13. A cross edge is any other edge.

If a vertex \(v\) is white when the edge \((u, v)\) is first explored in DF search, then the edge is a tree edge.

If a vertex \(v\) is grey when the edge \((u, v)\) is first explored in DF search, then the edge is a back edge.

If a vertex \(v\) is black when the edge \((u, v)\) is first explored in DF search, then the edge is either a forward or a cross edge.

Theorem 10.14. In a DF search of an undirected graph, every edge is either a tree edge or a back edge.
10.6 Topological Sort

**Definition 10.15.** A partial order on a set \( S \) is a relation \((\text{written } \leq)\) such that

1. the relation is reflexive: for every \( a, b \in S \) we have \( a \leq b \);

2. the relation is transitive: for every \( a, b, c \in S \) for which we have \( a \leq b \) and \( b \leq c \), then we have \( a \leq c \);

3. the relation is antisymmetric: if for \( a, b \in S \) we have \( a \leq b \) and \( b \leq a \), then we have \( a = b \).

**Examples**

Numbers (integers, rationals, reals) and \( \leq \) with its usual meaning.

Square matrices ordered by determinant.

Precedence operations in any computation.

We frequently want to linearize a partial order.

**Definition 10.16.** A topological sort or topological ordering of the vertices of a DAG is a linear ordering of the vertices such that if \((u, v)\) is an edge then \( \text{order}(u) \leq \text{order}(v) \).

**Theorem 10.17.** A directed graph is acyclic if and only if a DF search yields no back edges.

*Proof.* \( \Rightarrow \): Suppose there exists a back edge \((u,v)\). Then \( v \) is an ancestor of \( u \) in the DF forest. This implies that a path exists from \( v \) to \( u \) and that the back edge completes a cycle.

\( \Leftarrow \): Suppose that a graph \( G \) contains a cycle \( c \). Let \( v \) be the first vertex in \( c \) that is discovered during DFS, and let \((u,v)\) be the edge in \( c \) into \( v \). Then
at time $firsttime(v)$ there is a path of white vertices from $v$ to $u$, and by the white path theorem $u$ becomes a descendant of $v$ in the DF forest. This implies that $(u, v)$ must be a back edge.  \hfill \Box
TOP-SORT to produce a topological ordering
1 call DFS to compute finishing times lasttime(u) for each node u
2 as each vertex is finished, insert it on the front of a linked list
3 return the linked list

**Theorem 10.18.** The TOP-SORT algorithm produces a topological sort of a DAG.
10.7 Strongly Connected Components

**Theorem 10.19.** For an undirected graph \( G \), DFS can be used to identify connected components in the graph.

*Proof. Exercise* □

Directed graphs (and strongly connected components) are harder.

**Definition 10.20.** A strongly connected component (SCC) in a directed graph \( G = (V, E) \) is a maximal set \( U \subset V \) of vertices such that for any pair \( u, v \in U \) we have both a path in \( G \) from \( u \) to \( v \) and from \( v \) to \( u \).

**Definition 10.21.** The transpose \( G^T \) of a graph \( G = (V, E) \) is the graph \( G = (V, E^T) \) for which

\[
E^T = \{(v, u) : (u, v) \in E\}.
\]

Given an adjacency list for \( G \), it takes time \( O(|V| + |E|) \) to create \( G^T \) from \( G \).

The strongly connected components of \( G \) and of \( G^T \) are identical.

**Algorithm STRONGLY CONNECTED COMPONENTS**
1. call DFS for \( G \) to compute finishing times lasttime(u) for each vertex \( u \)
2. compute \( G^{-T} \)
3. call DFS for \( G^{-T} \), taking the vertices in decreasing order of lasttime(u)
4. output the vertices of each tree of step 3 as a different SCC

We will write \( u \sim v \) to mean that a path exists in the original graph \( G \) from vertex \( u \) to vertex \( v \).
Lemma 10.22. If two vertices are in the same SCC, then no path between them ever leaves the SCC.

Proof. Let $u$ and $v$ be any two vertices in the same SCC. Then there exist paths $u \rightsquigarrow v$ and $v \rightsquigarrow u$. Let $w$ be any other vertex on a path from $u$ to $v$, so we have $u \rightsquigarrow w \rightsquigarrow v \rightsquigarrow u$. Hence we have $u \rightsquigarrow w$ and $w \rightsquigarrow u$, so $w$ is in the same SCC as $u$. By symmetry, $w$ is also in the same SCC as $v$. □

Theorem 10.23. In any DF search, all vertices in the same SCC are in the same DF tree.

Proof. Let $r$ be the first vertex discovered in the SCC. Then all other vertices in the SCC are white at the time that $r$ is discovered. There are paths from $r$ to every other vertex in the SCC, and since no path leaves the SCC, all vertices on all such paths are white. By the white path theorem, all these vertices become descendants of $r$ in the DF tree. □

Definition 10.24. The forefather of a vertex $u$ in a graph is the vertex $w = \phi(u)$ such that a path exists from $u$ to $w$ and lasttime($w$) is maximal with this property.

- It is possible to have $\phi(u) = u$.
- lasttime($u$) ≤ lasttime($\phi(u)$)
- $\phi(\phi(u)) = \phi(u)$

Every SCC has a vertex that is the forefather of every vertex in the SCC. This is the first vertex in the SCC to be discovered and the last to be colored black.
Theorem 10.25. In an directed graph $G$, the forefather $\phi(u)$ of any node $u$ in any DF search of $G$ is an ancestor of $u$.

Proof. The theorem is trivially true if $\phi(u) = u$.

Assume $\phi(u) \neq u$. Consider the colors of the vertices at time $\text{firsttime}(u)$. We cannot have $\phi(u)$ black at this time, or else we would have $\text{lasttime}(\phi(u)) < \text{lasttime}(u)$, which would be a contradiction. If we were to have $\phi(u)$ grey at this time, then the theorem would be proved.

So we shall prove that $\phi(u)$ is not white at time $\text{firsttime}(u)$.

- If every intermediate vertex between $u$ and $\phi(u)$ is white at this time, then $\phi(u)$ becomes a descendant of $u$ by the white path theorem. This would then imply that $\text{lasttime}(\phi(u)) < \text{lasttime}(u)$, which would be a contradiction.

- If some intermediate vertex between $u$ and $\phi(u)$ is not white at this time, then let $t$ be the last nonwhite vertex on the path. The color of $t$ must be grey, since the successor of $t$ is white (by choice of $t$) and there is never an edge from a black to a white vertex. But this then says that there is a path of white vertices from $t$ to $\phi(u)$, and hence $\phi(u)$ is a descendant of $t$. From this we get that $\text{lasttime}(t) > \text{lasttime}(\phi(u))$, which contradicts the maximal time condition on the choice of $\phi(u)$.

\[\square\]

Corollary 10.26. In any DF search of a directed graph $G$, for any $u \in V$, vertices $u$ and $\phi(u)$ lie in the same SCC.
Proof. By the definition of $\phi(u)$ we have a path from $u$ to $\phi(u)$, and since $\phi(u)$ is an ancestor of $u$ we have a path from $\phi(u)$ to $u$. \qed

**Theorem 10.27.** In a directed graph $G = (V, E)$, two vertices $u, v \in V$ lie in the same SCC if and only if $\phi(u) = \phi(v)$ in a DF search of $G$.

Proof. $\iff$: If $u$ and $v$ are in the same SCC, then paths exist from $u$ to $v$ and from $v$ to $u$. By definition of $\phi(u)$ and $\phi(v)$, then, these are equal.

$
\rightarrow$: Assume that $\phi(u) = \phi(v)$. Then $u$ is in the same SCC as $\phi(u)$ and $v$ is in the same SCC as $\phi(v)$. Thus $u$ and $v$ are in the same SCC. \qed

**Theorem 10.28.** Algorithm SCC correctly computes the SCCs of a directed graph $G$.

Proof. We argue by induction on the number of DF trees found in the DFS of $G^T$ that the vertices of each tree form a distinct SCC. The inductive hypothesis is that under the assumption that all previous DF trees are SCCs, then the next tree found by Algorithm SCC is also a SCC. The induction begins with the first tree found, for which the hypothesis is trivially true.

Consider then a DF tree $T$ with root $r$ produced during the DFS of $G^T$. Let $C(r)$ be the set of vertices with forefather $r$:

$$C(r) = \{ v \in V : \phi(v) = r \}$$

We prove that a vertex $u$ is in $T$ if and only if $u \in C(r)$.

$\iff$: By Theorem ??, every vertex in $C(r)$ is placed in the same DF tree. Since $r \in C(r)$ and $r$ is the root of $T$, we have $C(r) \subset T$. 
\( \Rightarrow \): Let \( w \) be a vertex such that \( \text{lasttime}(\phi(w)) > \text{lasttime}(r) \). Inducting on the number of trees found, we have that \( w \) is not in \( T \), since at the time \( r \) is selected, \( w \) will already have been placed in the tree with root \( \phi(w) \).

In contrast, let \( w \) be a vertex such that \( \text{lasttime}(\phi(w)) < \text{lasttime}(r) \). Then \( w \) cannot be placed in \( T \); this would imply the existence of a path \( w \rightsquigarrow r \). And this would imply that

\[
\text{lasttime}(\phi(w)) \geq \text{lasttime}(\phi(r)) = \text{lasttime}(r),
\]

contradicting the assumption.

The tree \( T \) thus contains exactly those vertices \( u \) for which \( \phi(u) = r \). Thus, \( T = C(r) \), and we are done. \( \square \)