A complete graph on \( n \) nodes is a graph on \( n \) nodes for which all possible edges exist between pairs of

both \( v_i \) to \( v_j \) and \( v_j \) to \( v_i \). We refer to this as a strongly connected component.

If the graph is directed and paths exist of vertices \( v_i \) to \( v_j \) \( \in A \). If there exists a path in \( G \) connecting \( v_i \) and \( v_j \). Then \( A \subseteq G \). A connected component in a graph \( G \) is a maximal set of vertices \( A \subseteq G \) such that for every pair

A directed acyclic graph or DAG is a directed graph that has no cycles.

A cycle in a graph \( G \) is a path for which \( v_i = v_i \) where

\[ (v_1, v_2, \ldots, v_i, v_i, v_{i+1}, \ldots, v_n) \]

A path in a graph \( G \) is a sequence of edges \( (v_1, v_2, v_3, \ldots, v_n) \).

A subgraph is a graph \( G \) for which there is another graph \( G' \) \( A \subseteq G \) and for which we have

\( (G', A) = G \). For which there is another graph \( G' \) \( A \subseteq G \) and there is

\( \{ E \in (G', A) \} = G \). And there is

\( \{ (v_i, v_i) \} = G \). And there is

\( \{ u, \ldots, v_n \} = A \). A graph is a part \( C \) of \( G \). Where

Definitions

12.1 Graph Algorithms
The adjacency list representation for a graph on $n$ nodes consists of an $n$-long array of pointers to linked lists. The linked list pointed to from subscript $i$ consists of the subscripts of the nodes that are adjacent to $i$.

The adjacency matrix representation for a graph on $n$ nodes is a square $n \times n$ matrix whose entries $a_{ij}$ are 1 or 0 depending on whether there is or is not an arc (a directed arc) connecting vertex $i$ to $j$. If the graph is undirected, then this is a symmetric matrix. If the graph is directed, then in general this matrix will not be symmetric. One might use the $a_{ij}$ entries to store weights instead of just zeros and ones.

In general we do not think of the possibility that more than one distinct edge might connect a fixed pair of nodes. A clique in a graph $G$ is a subgraph $H$ of $G$ that is a complete graph.
- no "random" access for edge verification
- no search for edges and adjacent nodes
- exactly as many store nodes as edges
  - adjacency list
  - requires search on rows of columns to find edges
  - compressed format for dense graphs
  - adjacency matrix

spanning tree, minimal spanning tree, depth-first search, breadth-first search
Note that most graph algorithms have running times expressed in both $|V|$ and $|E|$.

- Construction of graphs/trees to match pairwise weights between nodes
- Any of the above problems with non-negative weights on the edges
- Find a Hamiltonian circuit that uses each vertex exactly once
- Find an Eulerian circuit that uses each edge exactly once
- Find a shortest path between all pairs of nodes
- Find all shortest paths from a given source to all other nodes
- Find the shortest path from a given source to a given sink
- Find the shortest path from a given source to a given sink
- Explore all nodes (breadth first and depth first)
Use an adjacency list for representation

Color unchanged to black when all adjacent vertices have been discovered

Color unchanged to grey when the vertices are discovered

All vertices start out white.

Keep three colors of vertices:

Start with a source vertex

12.3 Breadth-First Search
Time 5-7 take $\Theta(|V|)$ time. Loop 10-22 executes $\Theta(|E|)$ time. Queue and unqueue are constant.

```c
{ 22
    /* * queue = queue (queue) i.e. dequeue */
    color[n] = black
    { 21
        { 20
            /* * queue = queue \{v\} + i.e. enqueue */
            n = \{v\} + (\lambda) predecessor
            distance[n] = distance[n] + 1
            color[n] = (\lambda) gray
        } if (color[n] == (\lambda) white) 17
        for each \(v\) in adjacency(n) 16
            { 15
                head[queue] = n 14
            } } queue not empty 10
            { 9
                queue = \{s\}
                predecessor(n) = \{s\} 8
            } 7
            distance(n) = distance(n) + 1
            color[n] = white 6
        } for each vertex \(v\) in V 4
    } } \{v\} = (\lambda) infinity 3
```
Example: Consider a directed graph, the pathologic case when the inequality is very much not sharp.

\[ (a, n) \subseteq (a, s) \]

path from s to n followed by the edge (a, n).

Proof: If n can be reached from s then so can a. The shortest path from s to a can be no longer than the

\[ I + (n, s) \geq (a, s) \]

for any edge (n, a) ∈ E.

Lemma 12.2. Let \( (\Lambda, E) \) be any directed or undirected graph, and let \( s \in \Lambda \) be any vertex. Then

number of edges necessary in any path from u to v, or \( \infty \) if no path from u to v exists.

Definition 12.1. A shortest path distance \( g(u, v) \) from a node u to a node v is the minimum

12.4 Shortest Path Distances
Since we only queue nodes when they are white, and change the color to grey when they are queued, we have:

\[
(a, s)g' \leq 1 + (n, s)g' \leq 1 + (n)\text{distance} = (a)\text{distance}
\]

By the induction hypotheses, and then

\[
(n, s)g' \leq (n)\text{distance}
\]

Now when we discover a white vertex \( a \) when searching from node \( n \), we have for all other nodes \( v \),

\[
(a, s)g' \geq \infty = (a)\text{distance}
\]

and

\[
(s, s)g' = 0 = (s)\text{distance}
\]

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

\[
(a, s)g' \leq (n)\text{distance}
\]

Node \( n \in \Lambda \), we have

Lemma 12.3: Run BFS on a directed or undirected graph \( G \), from a source \( s \). Then for any
so the inequalities still hold.

\[
(1 + u_1 a) \text{distance} = 1 + (1 + u_1 a) \text{distance} \geq (u_1 a) \text{distance}
\]

and we have

\[
1 + (1 + u_1 a) \text{distance} = (1 + u_1 a) \text{distance}
\]

v1. So at that point we have

When we put vertex \( v = a + 1 \) onto the queue, we are at that point looking at the adjacency list for vertex

and everything stays ok.

\[
\text{distance} \geq 1 + (1 + u_1 a) \text{distance} \geq (u_1 a) \text{distance}
\]

If the head \( v_1 \) of the queue is taken off, we have

\[1.\quad \text{Proof. Induct on the number of queueing actions. When the queue holds only } v_1, \text{ the condition holds.} \]

for all \( i > 1 \).

\[
(1 + u_1 a) \text{distance} \geq (u_1 a) \text{distance}\]

and

\[
1 + (1 + u_1 a) \text{distance} \geq (u_1 a) \text{distance}
\]

Lemma 12.4. Suppose that during BFS, the queue contains the vertices \( (v_1, v_2, \ldots, v_k) \). Then
a is inserted into the queue

that for a \( \neq s \), \( \text{predecessor} \left( a \right) \) is set to \( n \) for some \( n \in \Lambda \).

that \( \text{distance} \left( a \right) \) is set to \( h \)

that a change in color to grey.

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

\[ \{ y = (a', s) : A \subseteq a \} = \neq \Lambda \]

Now let \( \Lambda \neq \emptyset \) be the set of vertices reachable at distance exactly \( k \), thus \( \Lambda \) is discovered.

vertices with finite distance values are discovered by the computation, and thus only reachable vertices are
discovered. Thus, the distance computation cannot set a finite distance value. Therefore only

Proof: Assume that a vertex \( v \) is unreachable from the start vertex \( s \). From Lemma 12.3 we have that

shortest path from \( s \) to \( \text{predecessor} \left( \Lambda \right) \) followed by the edge from \( \text{predecessor} \left( \Lambda \right) \) to \( v \).

for all other vertices. Further, for any non-source vertex \( v \), one of the shortest paths from \( s \) to \( v \) is the
discovers every vertex \( v \) that is reachable from the source and terminates with distance \( \lambda \).

Theorem 12.5: Given a directed or undirected graph on which we run BFS from a source \( s \). Then BFS
\( \text{predecessor}(a, n) \) and then following the edge from \( s \) to \( a \) can be made by using the shortest path from \( s \) to \( \text{predecessor}(a, n) \). This means that a shortest path

Finally, we observe that if \( a \notin \mathcal{V}^{-1} \), we have \( \text{predecessor}(a, n) \in \mathcal{V}^{-1} \). Thus proves the inductive step.

This proves the queue.

would imply that \( g(s, a) > L \). When \( a \) is discovered, \( \text{distance}(a, n) \) is set to \( L \). When this happens, \( a \) is discovered. \( s \) could not have been discovered earlier since that

\( \text{predecessor}(a, n) \) to \( n \), and \( n \) scanned by BFS. When this happens, \( a \) is discovered. \( s \) could not have been discovered earlier, since that

to \( \mathcal{V}^{-1} \), so must be on the queue and thus must at some point appear as the head of the queue and be

\( \text{distance}(a, n) \). Without loss of generality, we may assume that \( n \) is the first such vertex whose color is determined

since \( g(s, a) = L \), there must be a path from \( s \) to \( a \) of length \( L \) edges, and thus there must be a vertex

\( L \) is ever discovered \( (\text{until all the vertices of } \mathcal{V}^{-1} \text{ are on the queue).} \)

Now consider an arbitrary \( a \in \mathcal{V}^{-1} \). Then by Lemmas 12.2 and 12.3, we have that \( a \) is not discovered (if \( a \) \( \text{BFS terminates and that once } \text{distance}(a, n) \text{ and } \text{predecessor}(a, n) \) are set they are never changed. By Lemma 12.4, we have that the distances are nondecreasing.

The inductive hypotheses holds for \( k = 0 \) when \( A = 0 \). We now note that the queue is never empty until
Lemma 1.2.6. The BFS procedure constructs a predecessor subgraph that is a breadth-first tree.

The predecessor subgraph is a breadth-first tree if $P \subseteq \mathcal{G}$ contains all the vertices reachable from $s$ and

\[
\{\{s\} \setminus \Delta \in G : \exists (a, \text{predecessor}(a)) \} = \Delta
\]

and

\[
\{\forall \Delta \in G : \Delta \in \text{predecessor subgraph} \} \cap \{s\} = \Delta
\]

such that

\[
(d_{\mathcal{G}}(s, \text{predecessor subgraph} \mathcal{G})) = d_{\mathcal{G}}(s, \Delta)
\]
each edge, so the running time is $O(|V| + |E|)$.

Once again, we execute the recursive loop at most once for each edge, and in fact we do execute once for

```plaintext
14 return time + 1
17 time = time + 1
18 finishtime(n) = time
12 color(n) = black
11
10 { time = time + 1
8 if (color(n) == white)
6 (V) DFS-VISIT(n)
8 predecessor(n) = (V)
6 if (color(n) == white)
5 for each v in adjacency(n)
4 { time = time + 1
3 time = time + 1
2 disttime(n) = time
1 color(n) = grey
1 DFS-VISIT
}
```

Analysis:

```plaintext
14 DFS-VISIT(n)
10 if (color(n) == white)
9 { time = time + 1
7 for each vertex v in V(n)
6 time = 0
5 { time = time + 1
4 predecessor(n) = NULL
3 color(n) = white
2 for each vertex v in V(n)
1 Depth-First Search
```

Finishtime of the time when the color changes from grey to black.

We keep a timestamp `disttime` of the time when the color changes from white to grey and a timestamp
and that $\text{disctime}(n) > \text{finishtime}(a)$. Then we are in case (1). Case (2) is the symmetric case to (3) under the conditions $\text{disctime}(a) > \text{disctime}(n)$ and $\text{disctime}(a) > \text{disctime}(n)$. That is, we have that $\text{finishtime}(n) > \text{finishtime}(a)$, prior to $\text{finishtime}(n)$, processes from $a$ have been completed, and $a$ has been colored black. Prior to $\text{finishtime}(n)$, processes from a has been completed, and a has been colored already been explored prior to $\text{finishtime}(n)$, processes from $a$ have been discovered while $n$ was still grey, and hence $a$ is a descendant of $n$. Further, all the outgoing edges from $a$ have been discovered while $n$ was still grey, and hence $a$ is a descendant of $n$. Then a was a direct descendant of $n$ in the DFS tree.

Proof. Assume that $\text{disctime}(a) > \text{disctime}(n)$ and that $\text{disctime}(a) > \text{disctime}(n)$, then a was a direct descendant of $n$ in the DFS tree.

1. The intervals $(\text{disctime}(a), \text{finishtime}(a))$ and $(\text{disctime}(n), \text{finishtime}(n))$ are mutually disjoint.
2. The intervals $(\text{disctime}(a), \text{finishtime}(a))$ is entirely contained within (disctime(n), $\text{finishtime}(n)$).
3. The interval $(\text{disctime}(n), \text{finishtime}(n))$ is entirely contained within (disctime(a), $\text{finishtime}(a)$).
4. $a$ is a direct descendant of $n$ in the DFS tree.

5. The interval $(\text{disctime}(n), \text{finishtime}(n))$ is entirely contained within the interval $(\text{disctime}(a), \text{finishtime}(a))$.

6. The interval $(\text{disctime}(n), \text{finishtime}(n))$ is entirely contained within the interval $(\text{disctime}(a), \text{finishtime}(a))$.

7. The interval $(\text{disctime}(n), \text{finishtime}(n))$ is entirely contained within the interval $(\text{disctime}(a), \text{finishtime}(a))$.

then exactly one of the following holds:

Theorem 12.7. (Nested parentheses theorem) In any DFS of a directed or undirected graph and any
\[ \square \]

Within the interval \( \text{dist}(v) \), \( f(\text{finish}(v)) \). But then \( v \) is in fact a descendant of \( u \).

And hence the interval \( \text{dist}(v) \) is entirely contained in \( \text{dist}(u) \) by the previous theorem. Thus \( \text{dist}(v) > n \).

Further, \( \text{dist}(v) \) must be discovered before \( n \) is finished but after \( u \) is discovered. Thus, \( \text{dist}(v) > n \).

Ancestor, \( v \) is an ancestor, so by the previous theorem we have \( \text{finish}(v) > n \).

Then, since \( v \) is the first such vertex or an ancestor in the path to \( v \), let \( w \) be the predecessor of \( v \). Then, since \( w \) is the first such vertex, \( w \) is an ancestor of \( v \).

Assume that a white path (that is, a path consisting entirely of white vertices) exists from \( n \) to \( v \).

\[ \Rightarrow \]

Assume that a white path exists from \( n \) to \( v \). Let \( w \) be any vertex on the path from \( n \) to \( v \). Then \( w \) can be reached from \( n \) along a path consisting only of white vertices.

Theorem 12.9 (white path theorem). In a DF forest of a directed or undirected graph \( G \), a vertex \( u \) is a descendant of \( v \) if and only if at time \( \text{dist}(v) \) when \( v \) is discovered, \( v \) is a descendant of \( u \).

\[ \text{dist}(u) > \text{dist}(v) \]

Theorem 12.8. Vertex \( u \) is a proper descendant of \( v \) if \( u \) is in the \( D \) forest for a graph \( G \) and only if \( u \) is a descendant of \( v \) in the \( D \) forest.
Theorem 12.14. In a DFS search of an undirected graph, every edge is either a tree edge or a back edge.

Definition 12.13. A cross edge is any other edge.

Definition 12.12. A forward edge is a non-tree edge (v, w) connecting a vertex u to a descendant w.

Definition 12.11. A back edge is an edge (v, w) connecting a vertex u to an ancestor w.

First discovered by exploring edge (u, v).
Definition 12.16. A topological sort or topological ordering of the vertices of a DAG is a linear order of the vertices such that if \((u, v)\) is an edge then \(u\) precede \(v\) in the order.\(\)

We frequently want to linearize a partial order. Precedence operations in any computation. Square matrices ordered by determinant. Numbers (integers, rationals, reals) and \(\leq\) with its usual meaning.

Examples

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

4. \(\forall a \in S\)

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 \(a \leq c\).

1. The relation is reflexive: for every \(a \in S\) we have \(a \leq a\).

Definition 12.15. A partial order on a set \(S\) is a relation \((\subseteq\) such that
Theorem 12.18. The TOP-SORT algorithm produces a topological sort of a DAG.

1. Call DFS to produce a topological ordering.
2. As each vertex is finished, insert it on the front of a linked list.
3. Return the linked list.

□

and by the while path theorem $n$ becomes a descendent of $a$ in the DFS forest. This implies that $(a, n)$ must be a back edge.

and let $(n, a)$ be the edge in $c$ into $a$. Then at time distance$(n)$ there is a path of white vertices from $a$ to $n$.

Suppose there exists a back edge $(a, n)$, then $a$ is the first vertex in $c$ that is discovered during DFS.

Proof. Suppose there exists a back edge $(a, n)$, then $n$ is an ancestor of $a$ in the DFS forest. This implies that a path exists from $n$ to $a$ and that the back edge completes a cycle. A directed graph $G$ contains a cycle $c$ if and only if a DFS search yields no back edges.

Theorem 12.17. A directed graph is acyclic if and only if a DFS search yields no back edges.
The strongly connected components of $G$ and of $G'$ are identical.

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

$$\{ (n, a) : (n', a) \} = E$$

**Definition 12.21.** The transpose $G^T$ of a graph $G$ is the graph $G$ with the same vertices as $G$, but with the direction of each edge reversed.

**Definition 12.20.** A strongly connected component (SCC) in a directed graph $G$ is a maximal set of vertices that for any pair of vertices $u$ and $v$, there is a path from $u$ to $v$.

**Directly Connected Graphs (and Strongly Connected Components) are Harder.**

**Proof.** Exercise.

**Theorem 12.19.** For an undirected graph $G$, DFS can be used to identify connected components in $G$. The strongly connected components in $G$ are identical to the strongly connected components in the transpose of $G$.
Theorem 12.23. In a any DFS search, all vertices in the same SCC are in the same DFS tree.

Proof. Let v be the first vertex discovered in the SCC. Then all other vertices in the SCC are white at the time that v is discovered. There are paths from v to every other vertex in the SCC, and since no path leaves time that v is discovered, these vertices become white. By the White Path Lemma, all vertices on all such paths are white. By the White Path Lemma, all these vertices become white in the DFS tree.

Lemma 12.22. If two vertices are in the same SCC, then no path between them ever leaves the SCC.

We will write u ~ v to mean that a path exists in the original graph G from vertex u to vertex v.

Algorithm STRONGLY CONNECTED COMPONENTS

1. Call DFS for G to compute finishing times \text{finish}(u) for each vertex u
2. Compute G'-1
3. Call DFS for G'-1, taking the vertices in decreasing order of \text{finish}(u)
4. Output the vertices of each tree of step 3 as a different SCC

\[ \text{vertices in } G' \text{ with } \text{finish}(u) > \text{finish}(v) \text{ are in the same SCC} \]
(n)φ is not white at time disc$\text{on}(n)$. So we shall prove that φ \((n)φ\) is grey at this time. Therefore (n)φ (n)φ = φ, disc$\text{on}(n)φ$, (n)φ ≠ (n)φ, disc$\text{on}(n)φ$. Assume the theorem is not true. If we prove (n)φ (n)φ = φ, disc$\text{on}(n)φ$. Consider the colors of the vertices at time disc$\text{on}(n)$. If we cannot have two black at (n)φ (n)φ, disc$\text{on}(n)φ$. Assume (n)φ (n)φ, disc$\text{on}(n)φ$. n = (n)φ (n)φ, disc$\text{on}(n)φ$. Proof: The theorem is trivially true if n is an ancestor of n.

Theorem 12.25. In an directed graph G, the foreratherφ (n)φ of any node u in any DFS search of G is SCC to be discovered and the last to be colored black.

Every SCC has a vertex that is the forerather of every vertex in the SCC. This is the first vertex in the search that a path exists from u to w and finish$\text{on}(w)$ is maximal with this property.

Definition 12.24. The forerather φ (n)φ of a vertex u in a graph is the vertex w such that a path
a path from \( n \) to \( n' \).

**Proof.** By the definition of \( n \) we have a path from \( n \) to \( n' \), and since \( n \) is an ancestor of \( n' \) we have a path from \( n \) to \( n' \).

The same holds for \( n' \), which contradicts the maximal time condition on the choice of \( n \).

**Corollary 12.26.** In any DFS search of a directed graph \( G \), for any \( n, v \in V \), vertices \( n \) and \( v \) lie in the same SCC.

If some intermediate vertex between \( n \) and \( v \) 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 a path of white vertices from \( t \) 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 \( v \) which contradicts the maximal time condition on the choice of \( n \).

If every intermediate vertex between \( n \) and \( v \) is white at this time, then \( (n)ϕ<V(\text{finish\time} (t)) \) which would be a contradiction.
We prove that a vertex \( n \) is in \( T \) if and only if \( n \in \psi(C) \).

\[
\{ t = (a)\psi : \Lambda \in a \} = (t)\psi
\]

with \( \psi \).

Consider then a DFS tree \( T \) with root \( r \) produced during the DFS of \( G \). Let \( \psi(C) \) be the set of vertices found, for which the hypotheses is trivially true.

Then the next tree found by Algorithm SCC is also a SCC. The induction begins with the first tree being a distinct SCC. The inductive hypothesis is that under the assumption that all previous DFS trees form a distinct SCC. The inductive hypotheses is that vertices in the DFS of \( G \) that the vertices of each

**Theorem 12.28.** Algorithm SCC correctly computes the SCCs of a directed graph \( G \).

\( \square \)

**Proof.** We argue by induction on the number of DFS trees found in the DFS of \( G \). The vertices \( n \) and \( a \) are in the same SCC.

Thus \((a)\psi \) and \((n)\psi \) are in the same SCC as \((n)\psi \) and \((a)\psi \) are in the same SCC as \((n)\psi \). Then \( n \) is in the same SCC as \( a \) if and only if \( a \) is in the same SCC as \( n \). By definition of

\[
\forall n, a \in V, n \text{ and } a \text{ are in the same SCC } \iff \psi(n) = \psi(a)
\]

**Proof:** If \( n \) and \( a \) are in the same SCC, then \( (a)\psi \) and \((n)\psi \) are equal.

Thus \((a)\psi \) and \((n)\psi \) are in the same SCC as \((n)\psi \). Then \( n \) is in the same SCC as \( a \) if and only if \((n)\psi \) is in a DFS search of \( G \).
\[ (\mathcal{L}_n) \subseteq (\mathcal{L}_n) \subseteq (\mathcal{L}_n) \]

Thus, \( \mathcal{L} \subseteq \mathcal{L}_n \).

And this would imply that

\[ f_{\text{finish time}}(u) = (\mathcal{L}_n \cap \mathcal{L}_n) \subseteq (\mathcal{L}_n \cap \mathcal{L}_n) \]

In contrast, let \( u \) be a vertex such that \( f_{\text{finish time}}(u) > (\mathcal{L}_n \cap \mathcal{L}_n) \).

Then \( u \) cannot be placed with root \( \mathcal{L}_n \).

But \( u \) was placed with root \( \mathcal{L}_n \).

Hence, we have that \( u \) is not in \( \mathcal{L}_n \).

Since all the time \( t \) is detected, \( u \) will already have been placed in the tree.

So, \( u \) cannot be placed in \( \mathcal{L}_n \).

Therefore, we have that \( u \) is not in \( \mathcal{L}_n \).

And this would imply that \( f_{\text{finish time}}(u) > (\mathcal{L}_n \cap \mathcal{L}_n) \).

Induction on the number of trees.

Theorem 12.23: Every vertex in \( \mathcal{L}_n \) is placed in the same DF tree. Since \( t \in \mathcal{L}_n \), and \( u \) is the root of \( \mathcal{L}_n \), we have \( \mathcal{L}_n \subseteq \mathcal{L}_n \).

\[ \square \]