CSE 331: Introduction to Algorithm Analysis and Design

Graphs

1 Graph Definitions

Graph: A graph consists of a set of verticies V and a set of edges E such that:

$$G = (V, E)$$

$$V = \{v_0, v_1, \ldots, v_{n-1}\}$$

$$E = \{e_0, e_1, \ldots, e_{m-1}\} \subseteq V \times V$$

$$e_i = (u, v)$$

where $u \in V$ and $v \in V$. Note that $|V| = n$ and $|E| = m$.

Directed Graph: A graph is a directed graph if it can contain one-way edges without the same edge in the other direction.

$$(u, w) \in E \not\Longrightarrow (w, u) \in E$$

Undirected Graph: A graph is an undirected graph if every edge in the graph has a counterpart between the same nodes in the opposite direction.

$$(u, w) \in E \implies (w, u) \in E$$

By definition, every undirected graph is also a directed graph. Unless otherwise stated, for this course you can assume that $(u, u) \notin E$ and that graphs are undirected.

Path: A path P in a graph is a sequence of nodes that are all connected by edges.

$$P = (v_0, v_1, \ldots, v_{k-1})$$

$$v_i \in V$$

$$(v_i, v_{i+1}) \in E$$

Length of a Path: The length of a path P is the number of edges in the path. For

$$P = (v_0, v_1, \ldots, v_{k-1})$$

the length of P is $k-1$.

Cycle: A cycle is a path P the starts and ends at the same node.

$$P = (v_0, v_1, \ldots, v_{k-2}, v_0)$$

The minimum length of a cycle in an undirected graph is 3 (ie. no backtracking). The minimum length of a cycle in a directed graph is 2.

Simple Path: A path P is simple if no nodes are repeated. This is equivalent to saying there are no cycles in P. For and $v_i, v_j \in P$

$$i \neq j \implies v_i \neq v_j$$

Connected: Nodes v_i and v_j are connected if there exists a path from one node to the other. In directed graphs, the nodes are strongly connected if there exists a path from v_i and v_j and a path from v_j to v_i.

Connected Graph: A graph is connected if v_i and v_j are connected for all nodes $v_i, v_j \in V$.

Distance: The distance between connected nodes v_i and v_j is the length of the shortest path connecting v_i and v_j.

Tree: A tree is a connected graph that contains no cycles.

Forest: A graph is a forest if it has no cycles. A forest can be thought of as a set of trees. By definition, every tree is a forest.

2 Tree Proof

For an undirected graph G, any 2 of the following conditions implies the third.

1. G is connected
2. G has no cycles
3. G has $n-1$ edges

In this section, we will prove that properties 1 and 2 imply property 3. This is equivalent to showing that every tree $T = (V, E)$ has exactly $n-1$ edges.

Theorem 1. *Every undirected tree $T = (V, E)$ has exactly $n-1$ edges.*

Proof. We first setup the proof by adding some helpful descriptions of T without altering its structure. We first pick an arbitrary node $r \in V$ to be the root of T. We then orient each edge in T towards r (ie. each edge points up to the previous layer in the tree). The graph is undirected, but we describing it as directed to make the proof easier to describe.

Claim 1: Every non-root vertex u has exactly 1 outgoing edge.

We will prove this claim using contradiction by assuming that there exists a node u that doesn't have 1 outgoing edge. We will break this into 2 cases.

Case 1: If u has 0 outgoing edges, then u is not connected to the rest of T. Specifically, since we direct each edge towards r, the lack of an edge implies that there is no path from u to r. This contradicts the condition that T is connected.

Case 2: If u has 2 or more outgoing edges, then there is a cycle in T. Since the edges are directed towards r, we know that each outgoing edge leads to a node that is connected to r implying that the two nodes are connected. This creates a cycle by taking this path connecting the two node through r, then connecting the two nodes again through the two outgoing edges from u[1]. This contradicts the condition that T has no cycles.

Since both of these cases leads to a contradiction, every non-root node must have exactly 1 outgoing edge.

Claim 2: The root r has 0 outgoing edges.

Since the edges are directed towards r, there are no outgoing edges from r.

Claim 3: Every edge is an outgoing edge for exactly 1 non-root vertex.

This follows from the definition of an edge. An edge has exactly 1 source node and 1 termination node.

By combining claims 1 and 3 above, we can see that there is 1-1 correspondence between edges and non-root verticies. This follows since each non-root vertex has 1 outgoing edge and each edge is outgoing from 1 non-root vertex. There is one root with no outgoing edges so there are $n-1$ edges in T.

□

3 BFS Levels

Theorem 2. *If T is a BFS tree for $G = (V, E)$ and $x \in L_i$, $y \in L_j$, and $(x, y) \in E$, then i and j differ by at most 1 ($|i - j| \leq 1$).*

Proof. Without loss of generality (wlog), $i \leq j$. In other words, label the lesser of the two levels i. Since i and j are just labels, we can do this without losing the generality of the proof. This can be thought of as a shortcut which is equivalent to writing the same proof twice with i and j reversed. Instead, we relabel them and write the proof only once in terms of i being the smaller value (or equal).

This will be a proof by contradiction. We will assume that $i \not\leq j$ (or $i < j - 1$) and show that this leads to a contradiction. By the BFS algorithm we know that $x \in L_i$ and that when exploring L_{i+1}, edge (x, y) must be taken as no edges connecting the explored nodes to the unexplored nodes can be skipped. Since $i \leq j$, we know the algorithm explores x before, or at the same time, as y. By our assumption of $i < j - 1$, we know that $j > i + 1$ meaning that y is at least 2 levels below x. This implies that the algorithm skipped the edge (x, y) when exploring L_{i+1} which is a contradiction.

□

[1]Recall that this is an undirected graph and the direction of the edges is only used for describing them in the proof

Theorem 3. *If T is a BFS tree for $G = (V, E)$ rooted at node s and $x \in L_i$ then the shortest path from s to x contains i edges.*

Proof. Proof by induction on the number of edges in the path i.

Base case: For $i = 0$, the only node is the root s which needs no edges to get to itself.

Induction step: Assume that the shortest path from s to any node $u \in L_{i-1}$ contains $i - 1$ edges and prove that the shortest path from s to any node $v \in L_i$ contains i edges.

To prove this, we note that any such v must have an edge connecting it to a node $v' \in L_{i-1}$ in order for it to be explored by the BFS algorithm. By the inductive hypothesis, we know that the shortest path from s to v' contains $i - 1$ edges. By adding the edge (v', v) we have a path from s to v with i edges as desired. To finish the proof, we note that if there were a shorter path from s to v, then v would have been explored in a previous level. This is true by applying Theorem 2 to the shortest path from s to v. At each step along the path, the level increases by at most 1 which implies the shortest path contains at least i edges.

□

4 Connected Component Proof

We will now prove that the following $explore(s)$ algorithm computes the connected component of a node s in a graph $G = (V, E)$.

$explore(s)$:

1. $R = \{s\}$
2. while $\exists_{(u,w)\in E}(u \in R \wedge w \notin R)$
   (a) $R = R \bigcup \{w\}$
3. return R

This algorithm starts at s, as explores all nodes that are reachable from an already explored node.

Theorem 4. *The output R of $explore(s)$ is the connected component of s, $cc(s)$.*

Proof. For this theorem, we have to prove that $R = cc(s)$, both of which are sets. To prove set equality, we usually divide it into two separate proofs being $R \subseteq cc(s)$ and $cc(s) \subseteq R$. If both of these are true, then $R = cc(s)$ similar to $x \leq y$ and $y \leq x$ implying that $x = y$. Once we prove these two subset properties, we will have proven the theorem.

$\mathbf{R} \subseteq \mathbf{cc(s)}$: To prove this inequality, we will prove $w \in R \implies w \in cc(s)$ (every element in R is also in $cc(S)$). This is true by observing that an edge w is only added to R if there exist a path from s to w since the algorithm only explores a node by traveling along edges. Since there is a path from s to w, they are connected.

cc(s) $\subseteq$ R: Now we must prove $w \in cc(s) \implies w \in R$ which is the more difficult direction of this proof[2]. This will be a proof by contradiction of implication so we will assume that there is a node w in $cc(s)$ that is not in R ($w \in cc(s)$ and $w \notin R$).

Since $w \notin R$, we know that $w \in G \setminus R$. Also, since $w \in cc(s)$, there must exist a path p from s to w. Then there must exists an edge (x, y) along p that connects R to $G \setminus R$ such that $x \in R$ and $y \notin R$ which we call a crossing edge between the two subgraphs[3]. However, in *explore*, this edge must be explored in the while loop before termination which a contradiction of the algorithm.

We have proven that $R \subseteq cc(s)$ and $cc(s) \subseteq R$ which implies that $R = cc(s)$. Thus, $explore(s)$ outputs R which is the connected component of s.

□

5 Topological Ordering Proof

Define a topological ordering to be an ordering of all the nodes in a graph $u_0, u_1, \ldots, u_{n-1}$ such that $(u_i, u_j) \in E \implies i < j$.

Theorem 5. *If G has a topological ordering, then G is a directed acyclic graph (DAG).*

Proof. Proof by contradiction: Assume G is not a DAG, G has a directed cycle c. Let u_i be the node in c with the smallest index i (ie. u_i is the earliest node in the topological ordering of G. Then there must exists some edge $(u_j, u_i) \in c$ such that $j > i$ which is a contradiction of G having a topological ordering. Therefore, G must have no cycles and is a DAG.

□

[2]When proving set equality, it is common for one of the inequalities to much simpler to prove than the other

[3]It's possible that $x = s$ and/or $y = w$