Graph Review

1. Basics

A graph $G = (V, E)$ is defined as a set of vertices V (also called nodes) and a set of edges E. In an undirected graph, each edge is formally a size-2 subset of V; in a directed graph, it each edge is an ordered pair of vertices. We often

A **self-loop** (or just **loop**) is an edge (v, v). I.e., an edge from a vertex to itself.

A **multi-edge** is when we have more than one copy of the same edge, e.g. (u, v) and (u, v) both in the same graph. In a directed graph (u, v) and (v, u) are **not** a multi-edge (you have to have the same direction in a directed graph to be a multi-edge).

A graph is **simple** if it has no multi-edges and no self-loops.

We say an edge is **incident** to a vertex (or a vertex is incident to an edge) if the vertex is an endpoint of that edge. The **degree** of a vertex is the number of incident edges. In a directed graph the degree is the sum of the **indegree** (the number of edges coming in) and the **outdegree** (the number of edges leaving).

Two vertices are **neighbors** if there is an edge between them. In a directed graph, neighbors can be **in-neighbors** of u (all the vertices with an edge entering u) or **out-neighbors** of u (all the vertices with an edge leaving u).

2. Special Subgraphs

- A **walk** is a list of vertices $v_0, ..., v_k$ such that $(v_i, v_{i+1}) \in E$ for all $0 \leq i < k$.
- A **path** $v_0, ..., v_k$ is a walk that does not repeat a vertex.
- A **cycle** is a path $v_0, ..., v_k$ with $k \geq 1$ with an extra edge (v_k, v_0). The edge (v_k, v_0) must not appear on the path.

In a directed graph, the shortest possible cycle has 2 vertices; in a simple undirected graph, the shortest possible cycle has 3 vertices. A single vertex counts as a path (and a walk).

Do not confuse loops and cycles; they are distinct.

- An **independent set** is a subset of vertices with no edges between them.
- A **clique** (also called a **fully-connected** subgraph) is a set of vertices with every possible edge between them. I.e. $\forall u, v \in S(u \neq v \rightarrow (u, v) \in E)$

3. Connectivity

3.1. In undirected graphs

- A graph is **Disconnected** if there are vertices u, v such that there is no path from u to v.
- A graph is **Connected** if for all vertices u, v there is a path from u to v

For a connected graph, since the graph is undirected, there is also a path from v to u (it's the same path just in the reverse order). We could also have "walk" in the definitions above instead of "path" (the meaning of connected/disconnected stays the same).

A **connected component** (or just a **component**) is a *maximal* set of connected vertices. I.e., it is a set S such that

- For all $u, v \in S$ there is a path from u to v.
- For every T such that $S \subseteq T$, There are $u, v \in T$ such that there is not a path from u to v.

The connected components are the connected "pieces" of the graph.

3.2. In directed graphs

- A graph is **Weakly Connected** if the underlying undirected graph is connected. I.e., if ignoring the directions on the edges would give a connected (undirected) graph.
- A graph is **strongly Connected** if for all u, v there is a path from u to v (and from v to u).

Don't confuse "strongly connected" with "fully-connected!"

A **strongly connected component** is a *maximal* set of connected vertices. I.e., it is a set S such that

- For all $u, v \in S$ there is a path from u to v (and from v back to u.
- For every T such that $S \subseteq T$, There are $u, v \in T$ such that there is not a path from u to v.

3.3. Trees

A graph is **acyclic** if it has no cycles. An undirected, acyclic graph is called a **forest**. A forest that is also connected is called a **tree**[1].

4. Conventions

If we ask you about a "graph" you should assume that it is simple. When we ask you to give an example (or counter-example) of a graph, you must give a simple one unless we say otherwise.

If you are designing an algorithm from scratch, it may be to your benefit to make it a non-simple graph, but you need to describe clearly that it is not simple, and (if you use any library functions) how they will handle multi-edges (and/or loops).

We will sometimes expect you to figure out from context whether a graph is directed or undirected, based on the graph problems we're studying. For example, we will talk about coloring only in undirected graphs (because the direction of the edges don't matter).

[1] because each of the components of the forest is a tree! It's a pun. Not a good one, but it's what we've got.