Properties of DFS

→ **Proposition**: let G be an undirected graph on which a DFS traversal starting at vertex s has been performed. Then the traversal visits all vertices in connected component of s, and discovery edges form a spanning tree of the connected component of s.

→ **Proposition**: let $\vec{G}$ be a directed graph. DFS ~~search~~ on $\vec{G}$ starting at vertex s visits all the vertices of $\vec{G}$ that are reachable from s. Also, the DFS tree contains directed paths from s to every vertex reachable from s.

Running Time of DFS

→ DFS is called at most once on each vertex (since it gets marked as visited), and therefore every edge is examined at most twice for an undirected graph, once from each end of its vertices, and at most once in a directed graph, from its origin vertex.

If we let $n_s \leq n$ be the number of vertices reachable from a vertex s, and $m_s \leq m$ be the number of incident edges to these vertices, a DFS starting at s runs in $O(n_s+m_s)$ time, provided these conditions are met:

① graph is represented by a data structure such that creating and iterating through the incident-edges(v) takes $O(\deg(v))$ time

② we have a way to mark a vertex/edge explored

→ **Proposition**: let G be an undirected graph with n vertices and m edges. A DFS traversal of G can be performed in $O(n+m)$ time, & can be used to solve the following in $O(n+m)$ time:

- → Computing path btwn. two given vertices of G
- → testing if G is connected
- → computing spanning tree (if connected)
- → computing connected components
- → computing a cycle in G, or reporting that G has no cycles

→ Proposition: let $\vec{G}$ be a directed graph with n vertices and m edges. A DFS traversal of $\vec{G}$ can be performed in $O(n+m)$ time, can be used to solve:

- → computing directed path btwn. 2 given vertices
- → computing set of vertices of $\vec{G}$ that are reachable from vertex s
- → testing if G is strongly connected