Graphs
Undir: $m \leq \binom{n}{2}$, $\sum_v deg(v) = 2m$
Dir: $m \leq n * (n-1)$, $\sum_v indeg(v) = \sum_v outdeg(v) = m$
Subgraph: result of removing an edge, $V' \subseteq V, E' \subset E$
Induced SG: result of removing node. Is subgraph, $e \in E' \leftrightarrow (e \in E \wedge (u,v)) \in V$.
u connected to v: $(u \sim v) \to \exists$ path from u to v
G connected if $(u \sim v)$ $\forall (u,v)$

Representations
Adj. Matrix: row 1 = outgoing edge, col 1 = incoming edge
- Undir: $G = G^T$
- Find a neighbour: $O(n)$
- Access(v) = $O(1)$, traverse $V = O(n)$
- Traverse all edges of $v = O(n)$
- Traverse all edges of $G = O(n^2)$
- $O(V^2) = O(n^2)$ space

Adj. List: $\sum_v outdeg(v) = E$
- $O(V + E) = O(n + m)$ space
- Find a neighbour: $O(1)$
- Traverse all edges of $v = O(|neighbours(v)|)$
- Traverse all edges of $G = O(m)$
- Access(v) = $O(1)$, traverse $V = O(n)$

Trees
Tree: connected, acyclic graph
- Add edge $\to$ cycle; Remove edge $\to$ not connected
- n - 1 edges

Forest: graph with trees as connected components
Spanning tree: $T \subseteq G$ S.T $V(T) = V(G)$
Undir. is a tree $\leftrightarrow$ connected and E = V - 1

BFS
- $O(n + m)$ time (list), $O(n^2)$ (matrix)
- $\Theta(n)$ space for both list and matrix
- Finds all nodes, finds shortest path from s to all others

BFS
- $O(n + m)$ time (list), $O(n^2)$ (matrix)
- Finds all nodes, finds shortest path from s to all others

Edge Classes
The edge (u, v) refers to the FOREST, not the graph.
- Tree: $(u, v) \in$ Forest
- Forward: v is descendant of u
- Back: v is ancestory of u (back = fwd in undirs)
- Cross: none of the above are true.
- Und: DFS tree/fwd only, BFS tree/cross only
- Dir: DFS: all edges, BFS: no fwd edges

DAGs
- Source: no incoming edges
- Sink: no outgoing edges
- Multiple sources/sinks are possible. At least one of each in every DAG.
- Source & Sink $\to$ no cycles
- A Digraph is a DAG $\leftrightarrow$ DFS has no back edges

Toposort
- Produces ordering s.t $(u, v) \in E \to u$ appears before v in the ordering
- Digraph has a Toposort $\leftrightarrow$ it is a DAG
- Run DFS, order in decreasing order of finish time
- $O(n + m)$
- Orders are not unique.

SCCs
- Run DFS. Run DFS on G^T, in decreasing order of ftime. The forests of the transpose DFS are the SCCs.
- Independent of toposort ordering

Minimum Spanning Trees
- Spanning tree w/ minumum weight
- DFS and BFS build spanning trees, but *not* necessarily the MST.
- Optimal Substructure: if $T =$ MST of $G \to T[U]$, where $U \subset V$, if $T[U]$ is connected it is an MST.

Prim: add best vertex

Kruskal's Algorithm ($O((m + n)\log(n))$)
- Only consider edges that do NOT create a cycle (safe edges)
- Add lowest-cost edge on each iteration
- Has greedy-choice property

Edge Switching
- Let $e' \notin T$, where $T \cup \{e'\}$ has a cycle. For any $e \in E$, where e is in the cycle, $T \cup \{e'\} - \{e\}$ is a tree.

Single-Source Shortest Paths (SSSPs)
- The problem: find the min-cost path from source s to each vertex v.
- Shortest path is at most of length $n - 1$.

Dijkstra
- List + binary heap (as a min-PQ): $O((m + n)\log(n))$
- Matrix: $O(n^2 + m\log(n))$
- Fib heap: $\Theta(m + n\log(n))$
- No negative cycles (undir), no negative edges (dir)

Relaxing an Edge
- $d(v)$ only updated w/ relax $\to d(v) \geq d(s, v)$

relax(u, v): if $d(v) > d(u) + w(u, v)$:
    $d(v) \leftarrow d(u) + w(u, v)$

Bellman-Ford $O(VE)$
- Can handle negative edge weights (but not negative cycles!)

Procedure: Initialize $d(v) = \infty$, $\forall v$; set $d(s) = 0$; for $i \in \{1, \ldots, (n-1)\}$, relax *all edges*; for $e \in E$, if $d(u) + w(u, v) < d(v)$, return False. Else, return true. This last step checks for negative cycles.
- Use a DP table, dim $= 0 : (n - 1) \times n$
- $d_i[v] \leftarrow \min(d_{i-1}[v], \min_{u \in Neigh(v)}[d_{i-1}[u] + w(u, v)])$

Floyd-Warshall $O(n^3)$
- Allows negative weights, no negative cycles.
- All-pairs shortest path (APSP)

Suppose $V = \{1, 2, \ldots, n\}$.
Then, $d[i, j, k] =$ distance of shortest path from i to j, such that all intermediate vertices $\in \{1, 2, \ldots, k\}$.
- Case 1: don't need vertex k: $d[i, j, k] = d[i, j, (k - 1)]$.
- Case 2: need vertex k: $d[i, j, k] = d[i, k, (k-1)] + d[k, j, (k-1)]$.
- So, $d[i, j, k] = \min($Case 1, Case 2$)$
- Base cases: $d[i, j, 0] = w(i, j)$, $d[i, i, k] = 0$.