- **DFS** (outputs connected components, topological sort on a DAG. You also have access to the pre and post arrays.)
    - Types of graphs: Unweighted/Undirected graphs, directed graphs, in particular - Directional Acyclic Graph (DAG)
        - DFS ccnum is only useful if the Graph is undirected. For directed graph, always use SCC.
    - Inputs: G(V, E) in adjacency list representation
    - Access to: pre[], post[], prev[], and visited[T/F] arrays shared between explore & DFS. (visited is all True at end of run so not helpful for tracking) #506
        - pre[] & post[] are the pre and post order numbers and are created by Explore (not returning these to DFS/but we have access to pre & post)
        - prev[] is array which provides the 'parent' vertex for backtracking purposes
    - Outputs:
        - Undirect G = Vertices labelled by connected component number (ccnum)
        - Directed G = list of subgraphs (1 for each subcomponent)
    - Runtime: O(|V|+|E|) or O(n+m)
    - Data structure: Stack
- **Explore** subroutine (used by DFS)
    - Types of graphs: Unweighted/undirected or directed,
        - Directional Acyclic Graph (DAG)
    - Inputs: G(V,E) and start vertex v in V
    - Access to:
        - previsited (prev[]) = arrays of vertices before a given vertices (but not used by Explore, needed for DFS)
        - ccnum[]
    - Outputs:
        - visited(u) set to true for all vertices u reachable from v. Array of all the nodes in the graph, with the ones reachable from s set to True.
    - Runtime: O(|V|+|E|) or O(n+m)
    - Data structure:
- **BFS**
    - Types of graphs: Unweighted/undirected or directed
    - Inputs: G=(V,E) and start v in V
    - Access to: prev(u) giving vertex preceding u in shortest path from v
    - Outputs: dist(u) set to shortest path between v and reachable vertex u, or infinity of not reachable
    - Runtume: O(|V|+|E|) or O(n+m)
    - Data structure: Queue

- **Dijkstra's algorithm** - finds the shortest distance from a source vertex to all other vertices and a path can be recovered backtracking over the pre labels.
    - Types of graphs: Weighted/undirected or directed graphs (no negative weights)
    - Inputs: G=(V,E), start v in V
    - Access to: prev(u) giving vertex preceding u in shortest path from v
    - Outputs: dist(u) set to shortest distance between v and reachable vertex u, or infinity of not reachable
    - Runtime: O((|V|+|E|) log |V|) or O((n+m) log n)
    - Data structure: Priority queue/ minimum priority queue
- **Bellman-Ford** (compute the shortest path from s to t (weights allowed to be negative)
    - Types of graphs: Weighted/directed or undirected (can have negative weights)
    - Inputs: G=(V,E), start vertex (s)
    - Access to: detect negative cycles by comparing T[n,.] to T[n-1,.]
    - Outputs: shortest path from v to all other vertices
    - Runtime: O((|V|*|E|) or O(nm)
    - Data structure:
- **Floyd-Warshall** to compute the shortest path from all nodes to all other nodes (neg weights ok)
    - Types of graphs: Weighted/directed or undirected (can have negative weights)
    - Inputs: G=(V,E)
    - Access to: detect negative cycles by checking diagonals T[n,i,i]
    - Outputs: shortest path from all vertices to all other vertices
    - Runtime: O((|V|^3) or O((n^3)
    - Data structure:
- **SCC**s (outputs strongly connected components, and the metagraph of connected components.) (Create reverse graph = G^R, run DFS on G^R, find sink vertices of G by ordering by decreasing post #, run DFS again on this list, which returns ccnum) finding the maximal set of SCC
    - Types of graphs: general directed graphs (info)
    - Inputs: G(V,E)
    - Access to: strongly connected components via ccnum(u) of first DFS run, and all other DFS outputs/structures
    - Outputs: metagraph that has to be a DAG (contains connected components from 2nd DFS run)
    - Runtime: O(|V|+|E|) or O(n+m)
    - Data structure:
- **Kruskal's** algorithm to find a Minimum Spanning Tree (MST) (negative weights ok)
    - Types of graphs: connected, undirected, weighted graphs
    - Inputs: A connected undirected *G=(V,E)* with edge weights *we*

  - Access to:
  - Outputs: A minimum spanning tree defined by the edges X
  - Runtime: O(|E| log|V|) or O(m log n)
  - Data structure: disjoint-set
- **Prim's** algorithm to find a Minimum Spanning Tree (MST) helpful site (negative weights ok)
  - Types of graphs: connected, undirected, weighted graphs
  - Inputs: A connected undirected *G=(V,E)* with edge weights *we*
  - Access to:
  - Outputs: A minimum spanning tree defined by the array prev[]
  - Runtime: O((|E| log|V|) or O(m log n) runtime explanation
  - Data structure: Binary heap
- **Ford-Fulkerson** greedy algorithm to find max flow on networks.
  - Types of graphs: directed graphs with capacity of edges
  - Inputs: G=(V,E) with flow capacity c, a source node s, and a sink node t
  - Access to: Can trivially create the final residual network with G, and the outputted flow. Something along the lines of: explanation
    1. We run FF on the flow network to get the max flow.
    2. We use this to construct the residual graph.
  - Outputs: max flow
  - Runtime: O(C * |E|) or O(C*m), where C is size of max flow
  - Data structure: inked list that stores edge capacity & queue to store augmenting paths
- **Edmonds-Karp** to find max flow on networks. (Identical to Ford–Fulkerson, except search order for finding augmenting path must be the shortest path (BFS for G with all edge weights = 1.) that has available capacity.)
  - Types of graphs: directed graphs with capacity of edges
  - Inputs: G=(V,E) with flow capacity c, a source node s, and a sink node t
  - Access to: Can trivially create the final residual network with G and the outputted flow. Something along the lines of:
    1. We run EK on the flow network to get the max flow.
    2. We use this to construct the residual graph.
  - Outputs: max flow
  - Runtime: O(|V||E|^{2}) or O(n m^2)
  - Data structure:
- Key Difference: Ford-Fulkerson uses the DFS and Edmonds-Karp uses BFS
- **2-SAT** which takes a Conjunctive Normal Form (CNF) with all clauses of size ≤ 2 and returns a satisfying assignment if it exists. (uses SCC)
  - Types of graphs:

- Inputs: CFN f
- Access to: truth assignment of variables
- Outputs: Boolean (T = satisfiable, F = unsatifiable)
- Runtime: O(|V|+|E|) or O(n+m)
- Data structure: