Depth first search
- Starting from vertex v
- Mark v as marked
- Select u as an unmarked node adjacent to v
- If no u, quit
- If u, begin depth first search from u
- When search from u quits, select another node from v
- Similar to preorder tree traversal

Breadth first search
- Starting from node v
- Identify all nodes adjacent to v
- Add these to the set
- Determine set of unvisited nodes which are adjacent to this set
- Add these to the set
- Continue until no new nodes are encountered

An Example

What would the visit orders for DFS(1), DFS(5), BFS(1), BFS(5) look like?

Unweighted Shortest Path
- Find the shortest path (measured by number of edges) from a designated vertex S to every vertex
- Simplified case of weighted shortest path

Algorithm
- Starting from node S
- Distance from S to S is 0, so label as 0
- Find all nodes which are distance 1 from S
- Label as distance 1
- Find all nodes which are distance 2 from S
- These are 1 step from those labeled 1
- This is precisely a breadth first search
Positive Weighted Shortest Path

- Length is sum of the edges costs on the path
- All edges have nonnegative cost
- Find shortest paths from some start vertex to all vertices
- similar process to unweighted case
- Dijkstra's Algorithm

Distance at each node \( v \) is shortest path distance from \( s \) to \( v \) using only known vertices as intermediates

An example of a Greedy Algorithm

Solve problem in stages by doing what appears to be the best thing at each stage

Decision in one stage is not changed later

Ford Algorithm

\[
\text{FordAlgorithm}(G, s) \\
\text{for all vertices } v \text{ do} \\
\quad \text{dist}[v] = \infty; \ p[v] = \text{NULL}; \\
\quad \text{dist}[s] = 0; \\
\text{while there is } (v,u) \text{ that dist}[u] > \text{dist}[v] + \text{weight}(v,u) \text{ do} \\
\quad \text{dist}[u] = \text{dist}[v] + \text{weight}(v,u); \\
\quad p[u] = v; \\
\text{endwhile}
\]

Spanning tree

- subgraph of \( G \)
- contains all vertices of \( G \)
- connected graph with no cycles

Minimum spanning tree

- spanning tree with minimum cost
- only exists if \( G \) is connected
- number of edges is \( |V| - 1 \)
- two greedy methods
  - Kruskal's algorithm
  - Prim's algorithm
- differ in how next edge is selected
Kruskal's algorithm

- select edge with smallest weight as accept the edge if it does not cause a cycle
- determining if it causes a cycle: essentially the equivalence class (union/find) problem
- two vertices belong to the same set iff they are connected in the current spanning forest

Prim's algorithm

- grow the tree in successive stages
- in each stage, one node is picked as the root, we add an edge, and thus a vertex is added to the tree
- have a set on vertices in the tree and a set that is not in the tree

Prim's algorithm (cont.)

- at each stage, a new vertex to add to the tree is selected by choosing edge (u, v) such that the cost of (u, v) is the smallest among all edges where u is in the tree and v is not
- Build spanning tree starting from v1
- Result in the same spanning tree as that given by the Kruskal algorithm