Algorithm Name	Video on ECS60 Site	Dv = distance to vertex	ADT	Notes	Sample uses	General Steps	Big-Oh explanation
Critical Path Analysis Using Topological Sort	Video: 11-09 Time: 08:00	n/a	Queue	For graphs that are weighted, directed and acyclic. Schedule, dependency.	Building a house has many steps. Some tasks can be done simultaneously and some depend on others being done. Determine what tasks are most critical to do.	1. Convert Activity Node Graph (weighted verticies) to Event Node Graph (weighted edges) 2. Use Topological sort (O(V + E) which is very fast) to determine CRITICAL PATH 3. Go left to right to determine EARLIEST START times 4. Go right to left to determine LATEST COMPLETION times	O(V + E)
Unweighted Shortest Path	Video 11-09 Time: 30:25	1 + DISTw	Queue	For **unweighted** graphs (unbeatably fast big-Oh) Similar to **level-order-traversal** but done on a graph	Manhattan distances where all edges are the same	1. Use three column table: verticies, pv and dv 2. Start at distinguished vertex, get adjacent verticies, add to queue	O(V + E) because you are putting each vertex into the queue and are looking at every edge
Articulation Points	Video: 11-18 Time: 14:00	n/a	Stack of recursion	if a graph is not bi-connected, the verticies whose removal would disconnect the graph are **Articulation Points.**	A problem with bridges that act as bottlenecks	1. Pick starting vertex 2. Number verticies using DFS 3. low min: a. num(v) b. lowest low of children of v c. num of a backedge of v	O(V + E)
Dijkstra's	Video: 11-09 Time: 28:40 (he then talks about Breadth First Search so can skip to 36:00) Video: 11-13 Time: 13:00	COSTvw + DISTw cost of edge from v to w + distance to w (**cumulative**)	Min Heap, Min Heap w/Hash, or None.	For **weighted** graphs **Shortest path:** Given a **distinguished vertex** (so different from MST) and want to determine minimum path to all other verticies Cumulative is key - uses a min heap	There are a bunch of connected train stations and train cars scattered at the stations that need to go to a different station. You need to determine the best way to pick up and deliver the cars to their destinations.	1. Use four column table: verticies, known, pv and dv 2. Start at distinguished vertex, update adjacent verticies	O(V^2) using **no ADT** for **dense graphs** because E log E for a dense graph is (v^2)/2 log (v^2)/2 which is worse. O(E Log E) using heap. O(E Log V) technically 2(E log V) using heap w/hash
Prim's	Video: 11-13 Time: 36:30	COSTvw cost of edge from v to w (**non-cumulative**)	Same as Dijkstras	- Builds **MST** (note that MST uses an arbitrary starting vertex, unlike Dijkstra which is used for shortest path so uses a distinguished vertex) - Greedy Algorithm - **same as Dijkstra's** except **non-cumulative**	You want to make a network of computers so which paths do you choose so that your network has minimum of materials being used?	1. Choose ARBITRARY vertex 2. Select closest vertex	
Kruskal's	Video: 11-16 Time: 06:00	n/a	Find Union	- Build **disjoint sets** of **MST** and combine them. - Choose either Kruskal or Prim's based on Big-Oh		1. Sort all edges by weights from low to high 2. Choose union by height or union by size 3. Accept or reject 4. **stop** when **V-1 edges** completed	O(E log E) because it is O(E log V) to accept and reject all edges and O(E log E) to sort all edges
NetWork Flow (Ford-Fulkerson)	Video: 11-16 Time: 28:30	minimum(Cvw, Dw) minimum of capacity of edges from v to w and the flow that was into w itself	Max Heap, Max Heap w/Hash, or None.	- Greedy Algorithm but self correcting - Dijkstra-esque but uses a max heap instead of a min heap because we want maximum flow	You have a network of pipes and you want to know the max amount of water that can flow from the source to the sink	1. Use Dijkstra table with max heap to find augmented path with max flow 2. Done when sink is true	max flow * dijkstra big-oh