

<p>Graphs</p> <p>Undir: $m \leq \binom{n}{2}$, $\sum_v \deg(v) = 2m$</p> <p>Dir: $m \leq n * (n - 1)$, $\sum_v \text{indeg}(v) = \sum_v \text{outdeg}(v) = m$</p> <p>Subgraph: result of removing an edge, $V' \subseteq V, E' \subseteq E$</p> <p>Induced SG: result of removing node. Is subgraph, $e \in E' \leftrightarrow (e \in E \wedge (u, v)) \in V$.</p> <p>$u$ connected to v: $(u \sim v) \rightarrow \exists$ path from u to v</p> <p>G connected if $(u \sim v) \forall (u, v)$</p> <p>Representations</p> <p>Adj. Matrix: row 1 = outgoing edge, col 1 = incoming edge</p> <ul style="list-style-type: none"> - 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 <p>Adj. List: $\sum_v \text{outdeg}(v) = E$</p> <ul style="list-style-type: none"> - $O(V + E) = O(n + m)$ space - Find a neighbour: $O(1)$ - Traverse all edges of $v = O(\text{neighbours}(v))$ - Traverse all edges of $G = O(m)$ - Access(v) = $O(1)$, traverse $V = O(n)$ <p>Trees</p> <p>Tree: connected, acyclic graph</p> <ul style="list-style-type: none"> - Add edge \rightarrow cycle; Remove edge \rightarrow not connected - n - 1 edges <p>Forest: graph with trees as connected components</p> <p>Spanning tree: $T \subseteq G$ S.T $V(T) = V(G)$</p> <p>Undir. is a tree \leftrightarrow connected and $E = V - 1$</p> <p>BFS</p> <ul style="list-style-type: none"> - $O(n + m)$ 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 <p>BFS</p> <ul style="list-style-type: none"> - $O(n + m)$ time (list), $O(n^2)$ (matrix) - Finds all nodes, finds shortest path from s to all others <p>Edge Classes</p> <p>The edge (u, v) refers to the FOREST, not the graph.</p> <ul style="list-style-type: none"> - Tree: $(u, v) \in \text{Forest}$ - Forward: v is descendant of u - Back: v is ancestry 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 <p>DAGs</p> <ul style="list-style-type: none"> - Source: no incoming edges - Sink: no outgoing edges - Multiple sources/sinks are possible. At least one of each in every DAG. - Source & Sink \rightarrow no cycles - A Digraph is a DAG \leftrightarrow DFS has no back edges <p>Toposort</p> <ul style="list-style-type: none"> - Produces ordering s.t $(u, v) \in E \rightarrow u$ 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. <p>SCCs</p> <ul style="list-style-type: none"> - 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 	<p>Minimum Spanning Trees</p> <ul style="list-style-type: none"> - Spanning tree w/ minimum weight - DFS and BFS build spanning trees, but <i>not</i> necessarily the MST. - Optimal Substructure: if $T = \text{MST of } G \rightarrow T[U]$, where $U \subseteq V$, if $T[U]$ is connected it is an MST. <p>Prim: add best vertex</p> <p>Kruskal's Algorithm ($O((m + n) \log(n))$)</p> <ul style="list-style-type: none"> - Only consider edges that do NOT create a cycle (safe edges) - Add lowest-cost edge on each iteration - Has greedy-choice property <p>Edge Switching</p> <ul style="list-style-type: none"> - 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. <p>Single-Source Shortest Paths (SSSPs)</p> <ul style="list-style-type: none"> - The problem: find the min-cost path from source s to each vertex v. - Shortest path is at most of length $n - 1$. <p>Dijkstra</p> <ul style="list-style-type: none"> - 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) <p>Relaxing an Edge</p> <ul style="list-style-type: none"> - $d(v)$ only updated w/ relax $\rightarrow d(v) \geq d(s, v)$ <p>relax(u, v): if $d(v) > d(u) + w(u, v)$:</p> $d(v) \leftarrow d(u) + w(u, v)$ <p>Bellman-Ford $O(VE)$</p> <ul style="list-style-type: none"> - Can handle negative edge weights (but not negative cycles!) <p>Procedure: Initialize $d(v) = \infty, \forall v$; set $d(s) = 0$; for $i \in \{1, \dots, (n - 1)\}$, relax <i>all edges</i>; for $e \in E$, if $d(u) + w(u, v) < d(v)$, return False. Else, return true. This last step checks for negative cycles.</p> <ul style="list-style-type: none"> - Use a DP table, dim = 0 : $(n - 1) \times n$ - $d_i[v] \leftarrow \min(d_{i-1}[v], \min_{u \in \text{Neigh}(v)} [d_{i-1}[u] + w(u, v)])$ <p>Floyd-Warshall $O(n^3)$</p> <ul style="list-style-type: none"> - Allows negative weights, no negative cycles. - All-pairs shortest path (APSP) <p>Suppose $V = \{1, 2, \dots, n\}$.</p> <p>Then, $d[i, j, k]$ = distance of shortest path from i to j, such that all intermediate vertices $\in \{1, 2, \dots, k\}$.</p> <ul style="list-style-type: none"> - 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, $d[i, j, k] = \min(\text{Case 1, Case 2})$ - Base cases: $d[i, j, 0] = w(i, j), d[i, i, k] = 0$.
--	--

<p>Graphs</p> <p>Undir: $m \leq \binom{n}{2}$, $\sum_v \deg(v) = 2m$</p> <p>Dir: $m \leq n * (n - 1)$, $\sum_v \text{indeg}(v) = \sum_v \text{outdeg}(v) = m$</p> <p>Subgraph: result of removing an edge, $V' \subseteq V, E' \subseteq E$</p> <p>Induced SG: result of removing node. Is subgraph, $e \in E' \leftrightarrow (e \in E \wedge (u, v)) \in V$.</p> <p>$u$ connected to v: $(u \sim v) \rightarrow \exists$ path from u to v</p> <p>G connected if $(u \sim v) \forall (u, v)$</p> <p>Representations</p> <p>Adj. Matrix: row 1 = outgoing edge, col 1 = incoming edge</p> <ul style="list-style-type: none"> - 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 <p>Adj. List: $\sum_v \text{outdeg}(v) = E$</p> <ul style="list-style-type: none"> - $O(V + E) = O(n + m)$ space - Find a neighbour: $O(1)$ - Traverse all edges of $v = O(\text{neighbours}(v))$ - Traverse all edges of $G = O(m)$ - Access(v) = $O(1)$, traverse $V = O(n)$ <p>Trees</p> <p>Tree: connected, acyclic graph</p> <ul style="list-style-type: none"> - Add edge \rightarrow cycle; Remove edge \rightarrow not connected - n - 1 edges <p>Forest: graph with trees as connected components</p> <p>Spanning tree: $T \subseteq G$ S.T $V(T) = V(G)$</p> <p>Undir. is a tree \leftrightarrow connected and $E = V - 1$</p> <p>BFS</p> <ul style="list-style-type: none"> - $O(n + m)$ 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 <p>BFS</p> <ul style="list-style-type: none"> - $O(n + m)$ time (list), $O(n^2)$ (matrix) - Finds all nodes, finds shortest path from s to all others <p>Edge Classes</p> <p>The edge (u, v) refers to the FOREST, not the graph.</p> <ul style="list-style-type: none"> - Tree: $(u, v) \in \text{Forest}$ - Forward: v is descendant of u - Back: v is ancestry 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 <p>DAGs</p> <ul style="list-style-type: none"> - Source: no incoming edges - Sink: no outgoing edges - Multiple sources/sinks are possible. At least one of each in every DAG. - Source & Sink \rightarrow no cycles - A Digraph is a DAG \leftrightarrow DFS has no back edges <p>Toposort</p> <ul style="list-style-type: none"> - Produces ordering s.t $(u, v) \in E \rightarrow u$ 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. <p>SCCs</p> <ul style="list-style-type: none"> - 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 	<p>Minimum Spanning Trees</p> <ul style="list-style-type: none"> - Spanning tree w/ minimum weight - DFS and BFS build spanning trees, but <i>not</i> necessarily the MST. - Optimal Substructure: if $T = \text{MST of } G \rightarrow T[U]$, where $U \subseteq V$, if $T[U]$ is connected it is an MST. <p>Prim: add best vertex</p> <p>Kruskal's Algorithm ($O((m + n) \log(n))$)</p> <ul style="list-style-type: none"> - Only consider edges that do NOT create a cycle (safe edges) - Add lowest-cost edge on each iteration - Has greedy-choice property <p>Edge Switching</p> <ul style="list-style-type: none"> - 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. <p>Single-Source Shortest Paths (SSSPs)</p> <ul style="list-style-type: none"> - The problem: find the min-cost path from source s to each vertex v. - Shortest path is at most of length $n - 1$. <p>Dijkstra</p> <ul style="list-style-type: none"> - 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) <p>Relaxing an Edge</p> <ul style="list-style-type: none"> - $d(v)$ only updated w/ relax $\rightarrow d(v) \geq d(s, v)$ <p>relax(u, v): if $d(v) > d(u) + w(u, v)$:</p> $d(v) \leftarrow d(u) + w(u, v)$ <p>Bellman-Ford $O(VE)$</p> <ul style="list-style-type: none"> - Can handle negative edge weights (but not negative cycles!) <p>Procedure: Initialize $d(v) = \infty, \forall v$; set $d(s) = 0$; for $i \in \{1, \dots, (n - 1)\}$, relax <i>all edges</i>; for $e \in E$, if $d(u) + w(u, v) < d(v)$, return False. Else, return true. This last step checks for negative cycles.</p> <ul style="list-style-type: none"> - Use a DP table, dim = 0 : $(n - 1) \times n$ - $d_i[v] \leftarrow \min(d_{i-1}[v], \min_{u \in \text{Neigh}(v)} [d_{i-1}[u] + w(u, v)])$ <p>Floyd-Warshall $O(n^3)$</p> <ul style="list-style-type: none"> - Allows negative weights, no negative cycles. - All-pairs shortest path (APSP) <p>Suppose $V = \{1, 2, \dots, n\}$.</p> <p>Then, $d[i, j, k]$ = distance of shortest path from i to j, such that all intermediate vertices $\in \{1, 2, \dots, k\}$.</p> <ul style="list-style-type: none"> - 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, $d[i, j, k] = \min(\text{Case 1, Case 2})$ - Base cases: $d[i, j, 0] = w(i, j), d[i, i, k] = 0$.
--	--