22 Elementary Graph Algorithms

22.1 Representations of graphs

Definition. *Representations of graphs*

1. ***Adjacency List***

   (a) *An array of $|V|$ lists, one for each vertex in V. For each $u \in V$, $Adj[u]$ contains all the vertices v such that $(u, v) \in E$ (i.e. all vertices adjacent to u in G)*

   (b) *compact for **sparse** graphs ($|E| << |V|^2$)*

   (c) *For **directed graph**, the sum of lengths of all adjacency list is $|E|$, since edge of form (u, v) is represented as having v appearing in $Adj[u]$. (i.e. $u \to v$)*

   (d) *For **undirected graph**, the sum of lengths of all the adjacency lists is $2|E|$, since if (u, v) is an undirected edge, then u appears in v's adjacency list and vice versa*

   (e) *Memory: $\Theta(V + E)$*

   (f) *Search: $\Theta(E)$ Have no quick way of determining if a given edge (u, v) is present in the graph than to search for v in the adjacency list $Adj[u]$ (have to go through the list)*

2. ***Adjacency Matrix***

   (a) *Assume vertices numbered $1, 2, \cdots, |V|$ arbitrarily. We have a $|V| \times |V|$ matrix $A = (a_{ij})$ such that*
   $$a_{ij} = \begin{cases} 1 & \textit{if } (i, j) \in E \\ 0 & \textit{otherwise} \end{cases}$$

   (b) *good for **dense** graphs ($|E| \ = |V|^2$) or if need to tell if there is an edge between two vertices quickly*

   (c) *Memory: $\Theta(V^2)$*

   (d) *Search: $\Theta(1)$*

BFS $\Theta(V + E)$

Lemma. *For Proof of correctness*

1. *Let $G = (V, E)$ be directed or undirected graph, let $s \in V$ be an arbitrary vertex, for any edge $(u, v) \in E$,*
   $$\delta(s, v) = \delta(s, u) + 1$$

   Proof. The shortest path from s to v cannot be longer than shortest path from s to u followed by (u, v), since otherwise we just take shortest path from s to v and (u, v) which will be a shorter path. □

2. *Upon termination, for each vertex $v \in V$, the value $v.d$ computed by BFS satisfies $v.d \geq \delta(s, v)$*

   Proof. Induction on the number of enqueue operations. Inductive hypothesis is that $v.d \geq \delta(s, v)$ for all $v \in V$. Before s enqueued, I.H. holds since $v.d = \infty \geq 0 = s.d = \delta(s, s)$. Now consider a white vertex v that is just being discovered and we we serach $Adj[u]$. I.H. implies $u.d \geq \delta(s, u)$ By assignment of $v.d = u.d + 1$, and previous lemma (since $u \to v$)

   $$v.d = u.d + 1 \geq \delta(s, u) + 1 \geq \delta(s, v)$$

   □

3. *suppose queue Q contains vertices $\langle v_1, \cdots, v_r \rangle$, where v_1 is the head of Q and v_r is the tail. Then $v_r.d \leq v_1.d + 1$ and $v_i.d \leq v_{i+1}$ for $i = 1, \cdots, r - 1$*

   Proof. Proof by induction on number of queue operations. Initially, queue contains s only, lemma holds. Now we prove in inductive step that lemma holds after both dequeuing and enqueuing a vertex. If head v_1 is dequeued, v_2 is the new head. By inductive hypothesis $v_1.d \leq v_2.d$. But then we have $v_r.d \leq v_1.d + 1 \leq v_2.d + 1$, the remaining inequalities remain unaffected, so lemma holds with after dequeu of v_1. During an enqueue, say v, it becomes v_{r+1}. At that time, we just moved u from the queue. By inductive hypothesis, the new head v_1 satisfies $v_1.d \geq u.d$. We have

   $$v_{r+1}.d = v.d = u.d + 1 \leq v_1.d + 1$$

   Now to prove inequalities holds, by I.H. $v_r.d \geq u.d + 1$ and so $v_r.d \leq u.d + 1 = v.d = v_{r+1}.d$ and the remaining inequalities remain unaffected. So lemma holds during enqueue □

Theorem. ***Correctness of BFS*** *Let $G = (V, E)$ be directed or undirected graph, suppose BFS is run on G given $s \in V$. during execution, BFS discovers every vertex $v \in V$ reachable from s and upon termination, $v.d = \delta(s, v)$ for all $v \in V$. Moreover, for any vertex $v \neq s$ reachable from s, one of the shortest paths from s to v is a shortest path from s to $v.\pi$ followed by $(v.\pi, v)$*

Definition. ***Breadth-first Tree*** *For graph $G = (V, E)$ with source s, a predecessor subgraph of G, $G_\pi = (V_\pi, E_\pi)$ where*

$$V_\pi = \{v \in V : v.\pi \neq NIL\} \qquad E_\pi = \{(v.\pi, v) : v \in V_\pi - \{s\}\}$$

The predecessor graph G_π is a Breadth first tree if V_π consists of vertices raechable from s, and for all $v \in V_\pi$, the subgraph G_π contains a unique simple path from s to v that is also the shortest path from s to v in G

Lemma. *procedure BFS constructs π such that predecessor graph $G_\pi = (V_\pi, E_\pi)$ is a breadth-first tree*

DFS $\Theta(V+E)$

Definition. ***Depth-first Tree*** *For graph $G=(V,E)$ with source s, a predecessor subgraph of G, $G_\pi = (V, E_\pi)$ where*

$$E_\pi = \{(v.\pi, v) : v \in V \text{ and } v.\pi \neq NIL\}$$

The predecessor subgraph of a depth-first search forms a ***Depth-first forest*** *comprisong several* ***Depth-first trees****. The edges in E_π are* ***tree edges*** *(Note how we are not restricting V since DFS will include vertices unreachable from source s)*

Proposition. ***Timestamping***

1. ***Timestamp*** *$v.d$ records when v first discovered ($WHITE \to GRAY$) and $v.f$ records when finishes examing $Adj[v]$ ($GRAY \to BLACK$)*
2. *Vertex u is $WHITE$ before time u,d, $GRAY$ between $u.d$ and $u.f$ ands $BLACK$ thereafter*
3. *vertex v is a descendent of u in Depth-first forest if and only if v is discovered during the time in which u is gray*

Theorem. ***Parenthesis theorem*** *In DFS of G, for any two vertices u and v, exactly one of following holds*

1. *$[u.d, u.f]$ and $[v.d, v.f]$ are entirely disjoint, then neither u nor v is a descendent of the other in depth-first forest*
2. *$[u.d, u.f]$ contained entirely within $[v.d, v.f]$, and u is a descendent of v in the depth-first tree*
3. *$[v.d, v.f]$ is containedf entirely within $[u.d, u.f]$ and v is a descendent of u in depth-first tree*

Corollary. ***Nesting of descendents' interval*** *Vertex v is a proper descendent of u in depth-first forest for a graph G if and only if*

$$u.d < v.d < v.f < u.f$$

Theorem. ***White-Path Theorem*** *In depth-first forest of $G=(V,E)$, vertex v is a descendent of u if and only if at time $u.d$ that search discovers u, there is a path from u to v consisting entirely of white vertices*

Proposition. ***classification of edges***

1. ***Tree Edges*** *edges in depth-first forest G_π. (u,v) is a tree edge if v was first discovered by exploring edge (u,v)*

2. ***Back Edges*** *are* (u, v) *connecting* u *to an ancestor* v *in a depth-first tree. Self-loos (in directed graph) is also back edge*

3. ***Forward Edges*** *are edges* (u, v) *connecting* u *to a descendent* v *in a depth-first tree*

4. ***Cross Edges*** *are all other edges. They go between vertice same vertices in the same depth-first tree, as long as one vertex is not an ancestor of the other, or they can go between vertices in different depth-first trees.*

When (u, v) *is first explored, the color of vertex* v *tells us about its edge category*

1. $WHITE$ *indicate a tree edge*

2. $GRAY$ *indicate a back edge*

3. $BLACK$ *indicates a forward (if* $u.d < v.d$*) or cross edge (if* $u.d > v.d$*)*

Theorem. *In DFS of an undirected graph* G*, every edge of* G *is either a tree edge of a back edge*

23 Minimum Spanning Tree

Definition. ***The MST Problem*** *Given a connected, undirected, weighted graph* $G = (V, E)$*, with weight function* $w : E \to \mathbb{R}$*. Find an acyclic subset* $T \subseteq E$ *that connects all of the vertices and whose total weight*

$$w(T) = \sum_{(u,v)\in T} w(u, v)$$

is minimized. Since T *is acyclic and connects all of the vertices, we call it a* ***spanning tree****.*

Generic Greedy solution to MST Generic greedy method for MST involves maintaining a loop invariant on a set $A \subseteq E$,

Prior to each iteration, A is a subset of some MST

At each step, we determine an edge (u, v) that we can add to A without violating the invariant. Assume $A \subseteq E$ satisfies the loop invariant, a **safe edge** (u, v) is an edge such that $A \cup \{(u, v)\}$ maintains the invariant

Definition. ***Cut***

1. *A* ***cut*** $(S, V - S)$ *of an undirected graph* $G = (V, E)$ *is a partition of* V*.*

2. *An edge* $(u, v) \in E$ ***crosses*** *the cut if one of its endpoints is in* S *and the other is in* $V - S$

3. *A cut* ***respects*** *a set $A \subseteq E$ if no edges in A crosses the cut*

4. *An edge is a* ***light edge*** *crossing a cut if its weight is the minimum of any edge crossing the cut (maybe ≥ 1 light edges in case of tie)*

5. *A* ***light edge satisifying a given property*** *if its weight is minimum of any edge satisfying the property*

Proposition. 1. ***Possible Multiplicity*** *If there are n vertices in the graph, then each spanning tree has $n1$ edges.*

2. ***Cycle property*** *For any cycle C in the graph, if the weight of an edge e of C is larger than the individual weights of all other edges of C, then this edge cannot belong to a MST.*

Theorem. ***The greedy choice (light edge) is optimal (safe)*** *Let $G = (V, E)$ be connected, undirected graph with $w : E \to \mathbb{R}$. Let $A \subseteq E$ be included in some MST for G, let $(S, V - S)$ be any cut of G that respects A, and let (u, v) be a light edge crossing $C = (S, V - S)$. Then edge (u, v) is safe for A, i.e. $A \cup \{(u, v)\}$ is also in a subset of some MST*

Proof. Let T be a MST such that $A \subseteq T$. Assume T does not contain light edge (u, v), since otherwise $A \cup \{(u, v)\} \subseteq T$, done. Otherwise, $(u, v) \notin T$. Prove using cut-and-paste that (u, v) is safe. In the context of MST, inclusion of (u, v) in T forms a cycle, (u, v) along with path p, s.t. $u \overset{p}{\rightsquigarrow} v$. Since T is by definition simple, p is also simple. Since u and v are on opposite sides of the cut C, let (x, y) be an edge that crosses C. Note $(x, y) \notin A$ since C respects A. Let $T' = T \cup \{(u, v)\} \setminus \{(x, y)\}$. T is connected since removal of (x, y) breaks T into 2 components, and inclusion of (u, v) joins the components together. Now we show that T' is a MST. Since (u, v) is a light edge crossing C and (x, y) also crosses the cut, we have $w(u, v) \leq w(x, y)$, hence

$$w(T') = w(T) + w(u, v) - w(x, y) \leq w(T)$$

Since T already a MST, i.e. $w(T) \leq w(T')$, then $w(T) = w(T')$. T' is also MST. Now we show (u, v) is safe for A. since $A \subseteq T$, $A \subseteq T'$ since $(x, y) \notin A$. hence $A \cup \{(u, v)\} \subseteq T'$. Since T' is MST, (u, v) is safe for A. □

Corollary. ***Above theorem holds for cuts in form of connected component*** *Let $G = (V, E)$ be a connected, undirected, weighted graph. Let $A \subseteq E$ such that A is included in some MST of G. Let $C = (V_C, E_C)$ be a connected component in the forest $G_A = (V, A)$. If (u, v) is a light edge connecting C to some other component in G_A, then (u, v) is safe for A*

23.2 Kruskal and Prim's algorithms $O(E \lg V)$

Definition. ***Kruskal's algorithm*** *Finds a safe edge to the growing forest by finding, of all edges that connect any two trees in the forest, an edge (u, v) of least weight.*

1. ***Implementation*** *Needs a fast way to determine if an edge crosses connected components. Tracks trees in disjoint sets. Initializes vertices to disjoint sets with* MAKE-SET. *Sort edges by weight in nondecreasing order. Loop over all edges and include edge (u, v) to $A \subseteq E$ if u and v are not in the same set (evaluate with* FIND-SET*). Update disjoint sets with* UNION

2. ***Complexity*** *Assume disjoint-set-forest impl with union-by-rank and path-compression. Sorting takes $O(E \lg E)$. $O(E)$* FIND-SET *and* UNION *and $O(V)$* MAKE-SET *takes a total of $O((V + E)\alpha(V))$. Sicne G connected, $|E| \geq |V| - 1$, so disjoint-set operation takes $O(E\alpha(V)) = O(E \lg V) = O(E \lg E)$. In total, algorihtm takes $O(E \lg E)$. Note since $|E| < |V|^2$, $\lg |E| = O(\lg V)$, so running time is $O(E \lg V)$*

Definition. ***Prim's algorithm*** *Tree (A) starts from an arbitrary root vertex r and grows until the tree spans all vertices of V. Each step adds to the tree A a light edge that connects A to an isolated vertex, one on which no edge of A is incident. (so that the cut respects A)*

1. ***Implementation*** *Needs a fast way to select a new edge to add to tree. vertices not in the tree reside in a min-priority queue Q based on key attributes, where $v.key$ is the minimum weight of any edge connecting v to a vertex in the tree A. ($v.key = \infty$ if no such edge exists)*

2. ***Complexity*** *Assume Q impl with binary min-heap. building heap requires $O(\lg V)$ time. $O(V)$* EXTRACT-MIN *each taking $O(\lg V)$ amounts to $O(V \lg V)$. While loop iterates $O(E)$ times. The test for membership is $O(1)$ by keeping a bit in G for each vertex and tells if its not in Q and updating the bit once vertex is removed from Q.* DECREASE-KEY *taking $O(\lg V)$ each. Hence total time is $O(V \lg V + E \lg V) = O(E \lg V)$*

24 Single-Source Shortest Path

Definition. ***The Single-Paths Problem*** *Given a weighted, directedf graph $G = (V, E)$, with $w : E \to \mathbb{R}$.*

1. *The* ***weight of path*** *$w(p)$ for $p = \langle v_0, \cdots, v_k \rangle$ is given by*

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

2. *A* ***Shotest-path weight*** *$\delta(u, v)$ from u to v is given by*

$$\delta(u, v) = \begin{cases} min\{w(p) : u \overset{p}{\rightsquigarrow} v\} & \text{if there is a path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

3. *A* ***Shotest path*** *from u to v is defined as any path p with weight*

$$w(p) = \delta(u, v)$$

Definition. ***Variants***

1. ***Single-source shortest-path problem*** *Find the shortest path from a given* ***source*** *vertex $s \in V$ to each vertex $v \in V$*
2. ***Single-destination shortest-path problem*** *Find a shortest path to a given* ***destination*** *vertex t from each vertex $v \in V$. (By reversing direction of each edge, we can reduce this problem to a single-source problem)*
3. ***Single-Pair shortest-path problem*** *Find a shortest path from u to v for given vertices u and v. (If we solve single-source problem with source vertex u, we solve this problem also)*
4. ***All-pairs shortet-path problem*** *Find a shortest path from u to v for every pair of vertices u and v. (solving by single-source algo will be inefficient, there are better solutions)*

Proposition. ***Subpaths of shortest paths are shortest path (Optimal substructure)*** *Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$, let $p = \langle v_0, \cdots, v_k \rangle$ be a shortest path from vertex v_0 to vertex v_k and, for any i and j such that $0 \leq i \leq j \leq k$, let $p_{ij} = \langle v_i, v_{i+1}, \cdots, v_j \rangle$ be a subpath of p from vertex v_i to vertex v_j. Then p_{ij} is a shortest path from v_i to v_j*

Proposition. ***Cycles***

1. ***Negative-weight cycle***
   (a) *The shortest path cannot contain negative-weight cycles.*
   (b) *No path from s to a vertex on a cycle can be a shortest path, since we can always find a path with lower weight by following the proposed shortest path and then traversing the negative-weight cycle.*
   (c) *Hence we define for all $v \in C$ for some negative-weight cycle, $\delta(s, v) = \infty$*
2. ***Positive-weight cycle***
   (a) *The shortest path cannot contain positive-weight cycle,*
   (b) *since removing the cycle from the path produces a path with the same source and destination vertices and a lower path weight*
   (c) *For 0-weight cycles, we can always remove the cycle and get a shortest path without a cycle.*

(d) *Hence we assume shortest paths have no cycles (simple path). Since any acyclic path in G has at most $|V|$ distinct vertices, it contains at most $|V|-1$ edges, hence we try to find shortest path of at most $|V|-1$ edges*

Definition. ***Representing shortest path*** *Interested in predecessor subgraph $G_\pi = (V_\pi, E_\pi)$ where*

$$V_\pi = \{v \in V : v.\pi \neq NIL\} \qquad E_\pi = \{(v.\pi, v) : v \in V_\pi - \{s\}\}$$

Specifically, let $G = (V, E)$ be a weighted,directed graph with weight function $w : E \to \mathbb{R}$ and assume G contains no negative-weight cycles reachable from source vertex $s \in V$, so that shortest path are well-defined. A ***Shortest-paths tree*** *rooted at s is a directed subgraph $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$ such that*

1. *V' is the set of vertices reachable from s in G*
2. *G' forms a rooted tree with root s and*
3. *for all $v \in V'$,the unique simple path from s to v in G' is a shortest path from s to v in G*

Note shortest path or shortest path are not necessarily unique

Proposition. ***Shortest-path estimate and Relaxation***

1. ***Shortest-path estimate*** *For each $v \in V$, the shortest-path estimate $v.d$ is an upper bound on the weight of a shortest path from source s to v.*
2. ***Relaxation*** *The process of relaxing an edge (u, v) consists of testing whether we can improve the shortest path to v found so far by going through u and, if so, updating (improving) $v.d$ and $v.\pi$*
   (a) *Given $u.d$, $v.d$ and $w(u, v)$ for edge $u \to v$*
   (b) *Update $v.d$ if $u.d + w(u, v) < v.d$. In essence, take path along u instead of some other path*
   (c) *Only way to change $v.d$ and $v.\pi$*
3. ***Triangular Inequality (for weighted graphs)*** *For any edge $(u, v) \in E$ we have*

$$\delta(s, v) \leq \delta(s, u) + w(u, v)$$

Proof. Suppose p where $s \overset{p}{\rightsquigarrow} v$ is a shortest path, then claim holds by definition of shortest path. Otherwise, there is no shortest path from s to v. This implies that there is no shortest path to from s to u, since otherwise there exists shortest path p' such that $s \overset{p'}{\rightsquigarrow} u \to v$ which is a shortest path. Hence $\delta(s, v) = \delta(s, u)$ are either ∞ or $-\infty$. Hence the claim holds □

Proposition. ***Effect of relaxation on shortest-path estimates***

1. ***Upper-bound property*** *Let $G = (V, E)$ be weighted, directed graph with w. Let $s \in V$ be source vertex and graph initialized by* INITIALIZE-SINGLE-SOUCE(G, s) *then $v.d \geq \delta(s, v)$ for all $v \in V$ over any sequenece of relaxation steps on edges of G. Moreover, once $v.d$ achieves value $\delta(s, v)$, it never changes.*

   Proof. Prove by induction the claim $v.d \geq \delta(s, v)$ holds for all $v \in V$ on the number of relaxation steps. After initialization, $\infty = v.d \geq \delta(s, v)$ holds for all $v \in V \setminus \{s\}$, and since $s.d = 0 \geq \delta(s, s)$. For inductive step, we have that $x.d \geq \delta(s, x)$ for all $x \in V$. Assume we relax an edge (u, v), only $v.d$ will be changed

   $$v.d = u.d + w(u, v) \geq \delta(s, u) + w(u, v) \geq \delta(s, v)$$

   by I.H. and triangular inequality. In addition $v.d$ never change once $v.d = \delta(s, v)$. This is because $v.d$ never decreases as $v.d \geq \delta(s, v)$ holds for all $v \in V$ just proved and no operation increases $v.d$ □

2. ***No-path property*** *Given a weighted, directed graph $G = (V, E)$ with $w : E \to \mathbb{R}$ and there is no path from s to v. Then after graph is initialized by* INITIALIZE-SINGLE-SOURCE(G, s), *we have $v.d = \delta(s, v) = \infty$ and this equality is maintained as an invariant over any sequence of relaxation steps on edges of G*

   Proof. By definition of shortest path, $\delta(s, v) = \infty$ as there is no path from s to v. By upper-bound property $v.d \geq \delta(s, v) = \infty$, hence $v.d = \delta(s, v) = \infty$ □

   Lemma. *Let $(u, v) \in E$, then immediately after relaxing edge (u, v) by executing* RELAX(u, v, w), *we have $v.d \leq u.d + w(u, v)$*

   Proof. If $v.d > u.d + w(u, v)$, then by RELAX, $v.d = u.d + w(u, v)$. If $v.d \leq u.d + w(u, v)$, $v.d$ not updated. Hence $v.d \leq u.d + w(u, v)$ afterwards □

3. ***Convergence property (Given** $u.d = \delta(s, u)$**,** $v.d \xrightarrow{\textbf{Relax}(u,v)} \delta(s, v)$**)*** *Let $G = (V, E)$ be weighted, directed graph with weight function $w : E \to \mathbb{R}$, let $s \in V$ be source vertex, and let $s \rightsquigarrow u \to v$ is a shortest path in G for some $u, v \in V$. Suppose G is initialized by* INITIALIZE-SINGLE-SOURCE(G, s) *and then execute a sequence of relaxation steps that includes the call* RELAX(u, v, w) *on edges of G. If $u.d = \delta(s, u)$ at any time prior to relaxing edge (u, v), then $v.d = \delta(s, v)$ at all times afterwards*

   Proof. If $u.d = \delta(s, u)$ at any time prior to relaxing (u, v), then by upper-bound property, $u.d = \delta(s, u)$ stays the same. After relaxation on (u, v), by previous lemma

   $$v.d \leq u.d + w(u, v) = \delta(s, u) + w(u, v) = \delta(s, v)$$

the last equality given by optimal substructure of shortest path. By upper-bound property, $v.d \geq \delta(s, v)$, hence $v.d = \delta(s, v)$ and this property is maintained afterwards $\square$

4. ***Path-relaxation property*** *Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$ and let $s \in V$ be source vertex. Consider any shortest path $p = \langle v_0, \cdots v_k \rangle$ from $s = v_0$ to v_k. If G is initialized with* INITIALIZE-SINGLE-SOURCE(G, s) *and then a sequence of relaxation steps occurs that includes, in order, relaxing the edges (v_0, v_1), (v_1, v_2), $\cdots$, (v_{k-1}, v_k), then $v_k.d = \delta(s, v_k)$ This property holds regardless of any other relaxation steps that occur, even if they are intermixed with relaxations of the edges of p*

   Proof. Proof by induction $v_i.d = \delta(s, v_i)$ holds on i-th vertex in p relaxed. When $i = 0$, $v_0.d = s.d = 0 = \delta(s, s)$, by upper-bound property, the value never changes afterwards. In induction step, assume $v_{i-1}.d = \delta(s, v_{i-1})$. After relaxation of (v_{i-1}, v_i), $v_i.d = \delta(s, v_i)$ by convergence property and the equality is maintained thereafter by upper-bound property $\square$

Proposition. ***Relaxation and Shortest-paths tree***

1. *Let $G = (V, E)$ be a weighted, directed graph with $w : E \to \mathbb{R}$, let $s \in V$ be a source vertex, and assume G contains no negative-weight cycles that are reachable from s. Then after the graph is initialized with* INITIALIZE-SINGLE-SOURCE(G, s)*, the predecessor subgraph G_π forms a* ***rooted tree*** *with root s, and any sequence of relaxation steps on edges of G maintains this property as an invariant.*

   Proof. Proof consists of proving G_π is an acyclic graph by contradiction (i.e. assume there is a cycle and prove the cycle is in fact a negative weight cycle, which contradicts assumption of the problem). Then proving the graph is rooted at s, i.e. proving there is unique simple path from s to v in G_π $\square$

2. ***Predecessor-subgraph property*** *Let $G = (V, E)$ be a weighted, directed graph with $w : E \to \mathbb{R}$, let $s \in V$ be a source vertex, and assume G contains no negative-weight cycles that are reachable from s. Then after the graph is initialized with* INITIALIZE-SINGLE-SOURCE(G, s) *and execute any sequence of relaxation steps on edges of G that produces $v.d = \delta(s, v)$ for all $v \in V$, then the predecessor subgraph G_π is a shortest-path tree rooted at s.*

   Proof. Prove 3 properties of shortest-path trees given.

   (a) Prove V_π is the set of vertices reachable from s. Let $v \in V$ be not reachable from s, hence $\delta(s, v) = \infty$. Since $v.d$ and $v.\pi$ updated together in RELAX, implying $v.\pi = NIL$ and hence $v \notin V_\pi$

(b) Prove G_π forms a rooted tree with root s, follows from previous proposition

(c) Prove for all $v \in V_\pi$, the unique simple path $s \overset{p}{\leadsto} v$ in G_π is a shortest path from s to v in G. Let $p = \langle v_0, \cdots, v_k \rangle$ where $v_0 = s$ and $v_k = v$. For $i = 1, \cdots, k$, we have $v_i.d = \delta(s, v_i)$ (Path-Relaxation property) and $v_i.d \geq v_{i-1}.d + w(v_{i-1}, v_i)$, hence $w(v_{i-1}, v_i) \leq \delta(s, v_i) - \delta(s, v_{i-1})$. Summing weights along p

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_1) = \sum_{i=1}^{k} (\delta(s, v_i) - \delta(s, v_{i-1})) = \delta(s, v_k) - \delta(s, v_0) = \delta(s, v_k)$$

hence $w(p) = \delta(s, v_k)$ and thus p is a shortest path from s to $v = v_k$

□

24.2 Bellman-Ford algorithm $O(VE)$

Definition. *Bellman-Ford algorithm*

1. ***Goal*** *solves single-source shortest-paths problem in which edges may be negative*
   (a) *returns a boolean indicating whether or not there is a negative-weight cycle that is reachable from source*
   (b) *and the shortest path and their weight is no such cycle exists*
2. ***Implementation*** *works by progressively decreasing estimate $v.d$ until it achieves $\delta(s, v)$ by making $|V| - 1$ passes, where in each pass, every $v \in V$ is relaxed once.*
3. ***Runtime*** *$O(VE)$, initialization $\Theta(V)$, each $|V| - 1$ passes takes $\Theta(E)$ times (since relax every $e \in E$ requires traversing the entire adjacency list).*

Lemma. *Let $G = (V, E)$ be weighted, directed graph with source s and weight function $w : E \to \mathbb{R}$ asusme G contains no negatie-weight cycles that are reachable from s. Then after $|V| - 1$ iterations of for loop in the algorithm, we have $v.d = \delta(s, v)$ for all vertices v that are reachable from s*

Proof. Let $v \in V$ be arbitrary vertices reachable from s, let $p = \langle v_0 = s, \cdots, v_k = v$ be any shortest path from s to v. Since shortest path are simple, there are at most $|V| - 1$ edges. So $k \leq |V| - 1$. Since each of $|V| - 1$ iterations relax all $|E|$ edges, amongst them is the edge (v_{i-1}, v_i). By path-relaxation property, $v.d = v_k.d = \delta(s, v_k) = \delta(s, v)$ □

Corollary. *Let $G = (V, E)$ be weighted, directed graph with soruce s and weight function $w : E \to \mathbb{R}$ assume G contains no negative-weight cycles that are* ***reachable from*** *s. For each $v \in V$, there is a path from s to v if and only if* Bellman-Ford *terminates with $v.d < \infty$ when it is run on G*

Theorem. ***Correctness of Bellman-Ford Algorithm*** *Let* Bellman-Ford *be run on a weighted, directed graph $G = (V, E)$ with source s and weight $w : E \to \mathbb{R}$ If G contains no negative-weight cycles that are reachable from S, then the algorithm returns* True,*we have $v.d = \delta(s, v)$ for all vertices $v \in V$,and the predecessor subgraph G_π is a shortest-paths tree rooted at s. If G does contain a negative-weight cycle reachable from s, then the algorithm returns* False

Proof. Suppose G contains **no negative-weight cycles**. Prove $v.d = \delta(s, v)$ for all vertices $v \in V$. If v is reachable from s, prevous lemma proves the claim. Otherwise v not reachable from s, then claim follows from no-path property, i.e. $v.d = \delta(s, v) = \infty$. The predecessor subgraph property, along with the claim, implies G_π is shortest path tree. Now prove the algorithm returns True. At termination, for all $v \in V$

$$v.d = \delta(s, v) \le \delta(s, u) + w(u, v) = u.d + w(u, v)$$

so none of test for negative cycle in the algorithm returns False hence will return True. Suppose G has negative cycles reachable from s, let $c = \langle v_0, \cdots, v_k \rangle$, where $v_0 = v_k$, then

$$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$$

Prove by contradiction, if the algorithm returns True, then we have

$$v_i.d \le v_{i-1}.d + w(v_{i-1}, v_i)$$

for $i = 1, \cdots, k$. Summing equalities around the cycle

$$\sum_{i=1}^{k} v_i.d \le \sum_{i=1}^{k} (v_{i-1}.d + w(v_{i-1}, v_i)) = \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Note $\displaystyle\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i-1}.d$ since cycles hold the same vertices despite difference in the way they are indexed. Note $v_i.d$ is finite, so

$$\sum_{i=1}^{k} w(v_{i-1}, v_i) \ge 0$$

which contradicts the negative cycle assumption. □

24.3 Single-Source shortest paths in directed acyclic graphs $O(V + E)$

Definition. ***DAG Shortest-Path***

1. ***Motivation*** *Increase runtime by relaxing edges according to a topological sort of its vertices (so that we can use path-relaxation property and only relax every edge once)*

2. ***Implementation***

   (a) *Topologically sort the dag, i.e. if there is p such that $u \xrightarrow{p} v$, then u precedes v*

   (b) *Then make one pass over vertices in topologically sorted order. Relax each edge that leaves the vertex*

3. ***Runtime*** $O(V + E)$. *Topological sort takes* $\Theta(V + E)$ *time,* INITIALIZE-SINGLE-SOURCE *takes* $\Theta(V)$. *There is 1 pass where each edge is relaxed exactly once, each taking* $O(1)$, *hence amounts to* $\Theta(V + E)$

Theorem. *Correctness of DAG Shortest-Path algorithm* *If a weighted, directed graph $G = (V, E)$ has source vertex s and no cycles, then at termination of* DAG-SHORTEST-PATHS *procedure, $v.d = \delta(s, v)$ for all vertices $v \in V$, and the predecessor subgraph G_π is a shortest-path tree*

Proof. Show $v.d = \delta(s, v)$ for all $v \in V$ at termination. If v not reachable from s, $v.d = \delta(s, v) = \infty$ by no-path property. If v is reachable from s, then there is a shortest path $p = \langle v_0 = s, \cdots, v_k \rangle = v$. Because we process vertices in topological sorted order, the edges are relaxed in order. The path-relaxation property implies $v_i = \delta(s, v_i)$ at termination. The predecessor subgraph property implies G_π is a shortest path tree □

24.4 Dijstra's Algorithm $O(V^2)$ or $O(E \lg V)$

Definition. *Dijkstra's algorithm*

1. ***Use case*** *Solves single-source shortest-paths problem on a weighted, directed graph $G = (V, E)$ for the case in which all edges weights are nonnegative, i.e. $w(u, v) \geq 0$ for all $(u, v) \in E$*

2. ***Implementation***

   (a) *Maintains set S of vertices whose final shortest-path weights from s have already been determined.*

   (b) *Repeated selects a vertex $u \in V \setminus S = Q$, implemented with min-priority queue, with minimum shortest-path estimate.*

   (c) *Adds u to S*

   (d) *Relax all edges leaving u, i.e. $Adj[u]$*

3. ***Greedy*** *Since it chooses the lightest/closes vertex in $V \setminus S$ to add to set S*

4. ***Analysis Min-priority queue*** INSERT EXTRACT-MIN *called once per vertex, since each $u \in V$ added to S exactly once. The loop iterates $|E|$, size of adjacency list, and* DECREASE-KEY *is called at most once per loop (in* RELAX*). The runtime depends on how min-priority queue is implemented*

(a) ***Array*** INSERT *and* DECREASE-KEY $O(1)$, EXTRACT-MIN $O(V)$ *(have to go through entire array) total time* $O(V^2 + E) = O(V^2)$

(b) ***binary min-heap*** INSERT, DECREASE-KEY *and* EXTRACT-MIN *take* $O(\lg n)$. *Total runtime* $O((V + E)\lg V)$, *which is* $O(E \lg V)$ *if all vertices are reachable from source. Good if graph is sparce*

(c) ***Fibonacci heap*** $O(V \lg V + E)$

5. ***Comparison*** *Both Dijkstra's and Prim's algorithm uses a min-priority queue and grow the tree from source s, while updating other vertices*

Theorem. ***Correctness of Dijkstra's algorithm*** *Dijkstra's algorithm run on a weighted, directed graph* $G = (V, E)$ *with nonnegative weight* w *and source* s, *terminates with* $u.d = \delta(s, u)$ *for all vertices* $u \in V$.

Proposition. *The Loop invariant*

$$\textit{At the start of each iteration, } v.d = \delta(s, v) \textit{ for all vertex } v \in S$$

Its enough to show for each vertex $u \in V$, $u.d = \delta(s, u)$ *at time when* u *is added to the set, The upper-bound guarantees* $u.d = \delta(s, u)$ *holds afterwards*

Proof. Prove algo correct by proving invariant holds

1. **Initialization** Initially, $S = \emptyset$, hence invariant trivially true.

2. **Mainenance** Now we prove $u.d = \delta(s, u)$ for u added to S. Prove by contradiciton, let u be first vertex added for which $u.d \neq \delta(s, u)$ when it was added to the set. Note $u \neq s$ since s is first added with $s.d = \delta(s, s) = 0$, hence $S \neq \emptyset$. Also there must be some path connecting s to u, otherwise $u.d \neq \delta(s, u) = \infty$ which violates assumption that $u.d \neq \delta(s, u)$. If there is a path, there is a shortest path, let p be such path that connects $s \in S$ to $u \in V \setminus S$. Then at some point p crosses the cut $(S, V \setminus S)$. Let $y \in V \setminus S$ be the first vertices and x be y's parent, i.e. $y.\pi = x$. Now we decompose p

$$s \overset{p_1}{\rightsquigarrow} x \to y \overset{p_2}{\rightsquigarrow} u$$

Now we claim $y.d = \delta(s, y)$ when u is added to S. This is true because u is the first vertex added to S such that $u.d \neq \delta(s, u)$. Since $x \in S$, by I.H. we have $x.d = \delta(s, x)$ when x was added to S. Then (x, y) is relaxed at that time, hence the claim follows by convergence property. Now we obtain a contradiction, since y comes before u on a shortest path from s to u and all other edge in p_2 weights are non-negative, we have $\delta(s, y) \leq \delta(s, u)$, hence

$$y.d = \delta(s, y) \leq \delta(s, u) \leq u.d$$

But since both u and y is in $V \setminus S$ when u was chosen and we picked u instead of y hence $u.d \leq y.d$. The two inequalities yield a equality

$$y.d = \delta(s, y) = \delta(s, u) = u.d$$

Hence $\delta(s,u) = u.d$ contradicts the choice of u. Hence $u.d = \delta(s,u)$ when it was first added to S.

3. **Termination** At termination $Q = V \setminus S = \emptyset$, hence $S = V$, hence by previous invariant, $u.d = \delta(s,u)$ for all $u \in V$

$\square$

25 All-Pairs Shortest Path

Definition. ***All-Pairs shortest path***

1. ***Goal*** *Given $G = (V, E)$ with weight $w : E \to \mathbb{R}$. Find, for every pair $u, v \in V$, a shortest(least weight) path from u to v. Want to output in tabular form: each entry in u's row and v's column should be weight of a shortest path from u to v*

2. ***Naive solution*** *Run single-source shortest path algorithm $|V|$ times, once for each vertex as the source. Non-negative weight, use Dijkstra's algorithm, the min-heap impl of min-priority queue yields a runtime of $O(VE \lg V)$, fibonnaci heap impl yields runtime of $O(V^2 \lg V + VE) = O(V^3)$. If have non-negative weights have to use Bellman-Ford algorithm, with runtime of $O(V^2E)$, which is $O(V^4)$ if graph is dense.*

3. ***Representation of Graph*** *Use matrix representation. Assume vertices numbered $1, 2, \cdots, |V|$, an n-veretx directed $G = (V, E)$ is represented as a $n \times n$ matrix $W = (w_{ij})$ representing edge weights.*

$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ \textit{weight of directed edge (i,j)} & \text{if } i \neq j \wedge (i,j) \in E \\ \infty & \text{if } i \neq j \wedge (i,j) \notin E \end{cases}$$

The all-pairs shortest-path algorithm outputs $n \times n$ matrix $D = (d_{ij})$, where $d_{ij} = \delta(i,j)$. A ***Predecessor Matrix*** *$\Pi = (\pi_{ij})$, such that π_{ij} is NIL if either $i = j$ or there is no path from i to j, otherwise π_{ij} is predecessor of j on some shortest path from i.*

25.1 Shortest path and matrix multiplication with DP $O(V^3 \lg V)$

Definition. ***Shortest path and matrix multiplication***

1. ***Structure of shortest path*** *Given $W = (w_{ij})$, consider shortest path p from i to j, where p has m edges, assume no negative-weight cycles, and m is finite. If $i = j$, p has weight 0 and no edges. If $i \neq j$, then we can decompose p into*

$$i \overset{p'}{\leadsto} k \to j$$

where path p' now contains at most $m - 1$ edges, By optimal substructure of shortest path, p' is a shortest path from i to k, and so $\delta(i,j) = \delta(i,k) + w_{kj}$

2. ***Recursive solution*** *Let* $l_{ij}^{(m)}$ *be the minimum weight of any path from vertex* i *to vertex* j *that contains at most* m *edges. When* $m = 0$*, there is a shortest path from* i *to* j *with no edges if and only if* $i = j$*, hence*

$$l_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases} \qquad l_{ij}^{(m)} = Min\left\{ l_{ij}^{(m-1)}, \underset{1 \leq k \leq n}{Min}\{l_{ik}^{(m-1)} + w_{kj}\} \right\} = \underset{1 \leq k \leq n}{Min}\{l_{ik}^{(m-1)} + w_{kj}\}$$

The first term is the weight of a shortest path from i *to* j *in potentially* $m-1$ *edges, the latter is the minimum weight of paths, where all possible predecessor* k *of* j *is explored. The latter simplification is because* $w_{jj} = 0$*. The actual shortest-path weights are given by*

$$\delta(i,j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = \cdots$$

since a path from i *to* j *with more than* $n-1$ *edges is not simple anymore and hence cannot have a lower weight than a shortest path from* i *to* j *in under* $n-1$ *edges*

3. ***Bottom Up approach*** *The algorithm computes a series of matrices* $W = L^{(1)}, L^{(2)}, \cdots, L^{(n-1)}$ *for* $m = 1, \cdots, n-1$ *and* $L^{(m)} = (l_{ij}^{(m)})$ *and the final matrix* $L^{(n-1)}$ *contains the shortest path weights. Requires 3 nested for loop, hence runtime of* $O(n^3)$*. The procedure is very much similar to matrix multiplication, where*

$$c_{ij} = \sum_k a_{ij} \cdot b_{kj}$$

We have $L^{(m)} = L^{(m-1)} \cdot W$ *where* $\cdot$ *represent taking mins instead... The procedure* Extend-Shortest-Paths *is run* $n-1$ *times to yield* $L^{(n-1)}$ *hence the total runtime amounts to* $\Theta(n^4)$*.*

4. ***Improvement in runtime*** *To improve the runtime, we notice that the matrix operation is associative and hence we can compute* $L^{(n-1)}$ *in* $\lceil \lg(n-1) \rceil$ *by computing* $L^{(m)}$ *such that* m *is a power of 2. And once we loop to a point where* $m \geq n-1$*, we have* $L^{(m)} = L^{(n-1)}$ *as* $\delta(i,j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = \cdots$*. The total runtime is improved to* $O(n^3 \lg n) = O(V^3 \lg V)$*. The improvement lies in the fact that since there is no elaborate data structure, constant hidden in* Θ *is therefore small*

The FLoyd-Warshall algorithm $\Theta(V^3)$

Definition. ***Structure of a shortest path***

1. ***Concepts*** *Consider* ***intermediate vertices*** *of a shortest path* $p = \langle v_1, \cdots, v_l \rangle$ *is the set* $\{v_2, \cdots, v_{l-1}\}$

2. ***Observation*** *Assume* $V = \{1, 2, \cdots, n\}$ *For some subset* $\{1, 2, \cdots, k\} \subseteq V$*. Let* $i, j \in V$ *and* p *be a minimum-weight path from* i *to* j *with all intermediate vertices in* $\{2, \cdots, k-1\}$*.*

(a) *If k is not an intermediate vertex of p, The shortest path p with all intermediate vertices in $\{1, \cdots, k\}$ is also in $\{1, \cdots, k-1\}$*

(b) *If k is an intermediate vertex of p, then decompose p*

$$i \overset{p_1}{\rightsquigarrow} k \overset{p_2}{\rightsquigarrow} j$$

By optimal substructure of shortest path, p_1 is a shortest path from i to k with all intermediate vertices in $\{1, 2, \cdots, k\}$. Since k is not an intermediate vertex, all intermediate vertices of p_1 are in $\{1, 2, \cdots, k-1\}$. Hence

p_1 is a shortest path from i to k with all intermediate vertices in the set $\{1, 2, \cdots, k-1\}$; Similarly, p_2 is a shortest path from k to j with all intermediate vertices in the set $\{1, 2, \cdots, k-1\}$

3. ***Recursive solution*** *Let $d_{ij}^{(k)}$ be weight of a shortest path from i to j for which all intermediate vertices are in the set $\{1, 2, \cdots, k\}$. Note when $k = 1$, the set $\{1, 0\}$ has no intermediate vertex and includes i and j respectively and has one edge (i, j)*

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ Min\left\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right\} & \text{if } k \geq 1 \end{cases}$$

Hence $D^{(n)} = (d_{ij}^n)$ gives the right answer since all intermediate sets are in $\{1, \cdots, n\}$. So

$$d_{ij}^{(n)} = \delta(i, j) \quad \text{for all } i, j \in V$$

4. ***Bottom Up Approach*** *Runtime $O(V^3)$ because of the triple for loop, each taking $O(1)$ to look up previously computed values and calculate the minimum. Again, the code is tight, and so constat hidden in Θ notation is small*

5. ***Constructing shortest path*** Π

   (a) *from D of shortest path weights after computing D*

   (b) *at the same time D is calculated*

Definition. ***Transitive Closure of a directed graph*** *$G^* = (V, E^*)$ where*

$$E^* = \{(i, j) : \quad \text{there is a path from vertex } i \text{ to } j \text{ in } G\}$$

Solutions

1. *We can compute transitive closure by assign weight of 1 to each edge in E and run Floyd-Warshall algorithm. So if $d_{ij} < n$ there is a path from i to j otherwise $d_{ij} = \infty$*

2. *To save time and space we substitute logical operations land and lor with arithmetic operation in Floyd-Warshall algorithm Define t_{ij}^{k} be 1 if there is a path from i to j with all intermediate set in G and 0 otherwise.*

$$t_{ij}^{(0)} = \begin{cases} 0 & \text{if } i \neq j \text{ and } (i,j) \notin E \\ 1 & \text{if } i = j \text{ or } (i,j) \in E \end{cases} \qquad t_{ij} = t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)})$$

Then compute $T^{(k)} = (t_{ij}^{(k)})$ in order of increasing k (bottom up). The runtime is $\Theta(n^3)$, same as previous algorithm. But is quite faster and memory efficient since operates on bits (logical) instead of on integer words (arithmetic)

Maximum Flow

26.1 Flow Networks

Definition. *Flow networks*

1. ***Flow network*** *A flow network $G = (V, E)$ is a directed graph in which*
   (a) *each edge $(u, v) \in E$ has a nonnegative* ***capacity*** $c(u, v) \geq 0$.
   (b) *if $(u, v) \in E$, then the edge in reverse direction $(v, u) \notin E$*
   (c) *if $(u, v) \notin E$, then $c(u, v) = 0$*
   (d) *No self-loops*
   (e) ***source*** *s and a* ***sink*** *t*
   (f) *Assume each vertex lies on some path from s to t, i.e. for all $v \in V$, we have $s \rightsquigarrow v \rightsquigarrow t$*
   (g) *$|E| \geq |V| - 1$ since each vertex other than s has at least one entering edge*
2. ***Flow*** *Let $G = (V, E)$ be flow network with capacity function c. A flow in G is a real-valued funtion $f : V \times V \to \mathbb{R}$ satisfying*
   (a) ***Capacity Constraint*** *For all $u, v \in V$, we have $0 \leq f(u, v) \leq c(u, v)$*
   (b) ***Flow Conservation*** *For all $u \in V \setminus \{s, t\}$, we have*

$$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$$

If $(u, v) \notin E$, then no flow from u to v and $f(u, v) = 0$. Denote $f(u, v)$ the flow from vertex u to v. The ***value of $|f|$ of a flow*** *f is defined as difference between total flow out of source and total flow into sink*

$$|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s)$$

as a typical flow network does not have edges into source s we have

$$|f| = \sum_{v \in V} f(s, v)$$

3. ***Maximum-Flow problem*** *Given flow network G with source s and sink t, find a flow f of maximum value $|f|$*

4. ***Transformation to flow network***

   (a) ***Antiparallel edges*** *An antiparallel edge is the pair (v_1, v_2) and (v_2, v_1), which violates flow network. We can transform such graph into a flow network by taking one edge and decompose into 2 edges with an additional intermediate vertex, while set bot new edges' capacity constraint to the original edge. The two graphs are equivalent*

   (b) ***Multiple sources and sinks*** *Add a* ***supersource*** *s and directed edge (s, s_i) with capacity $c(c, c_i) = \infty$ for each $i = 1, \cdots n$ and likewise add a* ***supersink*** *t with directed edge $(t_i, t$ with capacity $c(t_i, t) = \infty$. In other words, provided unlimited flow as desired for multiple sources s_i and sinks t_i. The two graphs are equivalent*

26.2 The Ford-Fulkerson Method

Definition. ***General Steps***

1. *let $f(u, v) = 0$ for all $u, v \in V$*

2. *At each step, increase flow value in G by finding an* ***augmenting path*** *in an associated* ***residual network*** *G_f*

3. *Repeat until the residue network has no more augmenting paths*

Definition. ***Residual Network***

1. ***General Idea*** *G_f consists of edges with capacities that represent how we can change the flow on edges of G.*

   (a) *An edge (u, v) of G can admit $c(u, v) - f(u, v) = c_f(u, v)$ amount of additional flow (if edge has flow equal to capacity then $c_f(u, v) = 0$)*

   (b) *An edge (u, v) of G can also reduce their flow by an amount up to $f(u, v) = c_f(v, u)$. The edge (v, u) placed in G_f is able to admit flow in opposite direction to (u, v), at most cancelling out the flow on (u, v)*

2. ***Residual Capacity*** *Given flow network G and a flow f. Consider $u, v \in V$, the residual capacity $c_f(u,v)$ is defined by*

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(v,u) & \text{if } (v,u) \in E \\ 0 & \text{otherwise} \end{cases}$$

   Note since flow network disallows antiparallel edges, exactly one of the cases applies

3. ***Residual Network*** *Given flow network G and flow f, the residual network of G induced by f is $G_f = (V, E_f)$ where*

$$E_f = \{(u,v) \in V \times V : c_f(u,v) > 0\}$$

4. ***Residual Edge*** *Edges in residual network is called residual edge E_f, which can be either edges in E, their reversal, or both*

$$|E_f| \leq 2|E|$$

5. ***Augmentation*** *If f is a flow in G and f' is a flow in corresponding residual network G_f, then $f \uparrow f'$, the augmentation of flow f by f', to be a function $(f \uparrow f') : V \times V \to \mathbb{R}$*

$$(f \uparrow f')(u,v) = \begin{cases} f(u,v) + f'(u,v) - f'(v,u) & \text{if } (u,v) \in E \\ 0 & \text{otherwise} \end{cases}$$

   *The idea is we increase flow on $(u,v) \in V$ by $f'(u,v)$ but decrease it bey $f'(v,u)$ because pushing flow on reverse edge in residual network signifies decreasing the flow in the original network, this is called **cancellation***

   Lemma. *Let $G = (V,E)$ be flow network with source s and sink t, let f be a flow in G. Let G_f be residual network of G induced by f, and f' be a flow in G_f. Then $f \uparrow f'$ is a flow in G with value $|f \uparrow f'| = |f| + |f'|$*

Definition. ***Augmenting Paths (Improves value of flow)***

1. ***Augmenting Path*** *Given flow network G and a flow f, an augmenting path p is a simple path from s to t in the residual network G_f.*

2. ***Residual Capacity of an Augmenting Path*** *The maximum amount by which we can increase the flow on each edge in an augmenting path p the residual capacity of p (such that capacity constraint is satisifed in G)*

$$c_f(p) = Min\{c_f(u,v) : (u,v) \text{ is on } p\}$$

3. ***Flow of an Augmenting Path*** *We get a flow of an augmenting path p by assigning the residual capacity of p, i.e. $c_f(p)$, to every edge on the path p. Define function $f_p : V \times V \to \mathbb{R}$ by*

$$f_p(u,v) = \begin{cases} c_f(p) & \text{if } (u,v) \text{ is on } p \\ 0 & \text{otherwise} \end{cases}$$

   Lemma. *f_p is a flow in G_f with $|f_p| = c_f(p) > 0$*

4. *If we augment f by f_p, i.e. $f \uparrow f_p$, we get another flow in G whose value is closer to the maximum*

   Corollary. *Let $G = (V, E)$ be a flow network, let f be a flow in G, and let p be an augmenting path in G_f. Let f_p be defined as previously, then the function $f \uparrow f_p$ is a flow in G with value*

$$|f \uparrow f_p| = |f| + |f_p| > |f|$$

   Proof. Follows from $|f_p| = c_f(p) > 0$ and $|f \uparrow f'| = |f| + |f'|$ □

Definition. ***Cut of Flow Networks (Determines when max flow is found)***

1. ***Cut*** *A cut (S, T) of a flow network $G = (V, E)$ is a partition of V into S and $T = V \setminus S$ such that $s \in S$ and $t \in T$*

2. ***Net Flow across a cut*** *If f is a flow, then the net flow $f(S, T)$ across the cut (S, T) is defined to be*

$$f(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v) - \sum_{u \in S} \sum_{v \in T} f(v,u)$$

   *Note **value of flow** $|f|$ is the net flow across cut $(\{s\}, V \setminus \{s\})$*

3. ***Capacity of a cut*** *The capacity of cut (S, T) is*

$$c(S,T) = \sum_{u \in S} \sum_{v \in T} c(u,v)$$

   Note we only consider flow from S to T, ignoring edges in the reverse direction (different from flow which considers both directions)

4. ***Minimum Cut*** *The minimum cut of a network is a cut whose capacity is minimum over all cuts of the network*

Lemma. *Let f be a flow in a flow network G with source s and sink t, and let (S, T) be any cut of G. Then the net flow across (S, T) is $f(S, T) = |f|$*

Corollary. *The value of any flow f in a flow network G is bounded from above by the capacity of any cut of G. (Implies that optimal $|f|$ is minimum capacity of all cuts in G)*

Proof. Let (S, T) be any cut of G and f be any flow. By previous lemma we have

$$|f| = f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u) \leq \sum_{u \in S} \sum_{v \in T} f(u, v) \leq \sum_{u \in S} \sum_{v \in T} c(u, v) = c(S, T)$$

□

Theorem. ***Max-flow Min-cut theorem*** *If f is in a flow network $G = (V, E)$ with source s and sink t, then the following conditions are equivalent*

1. *f is a maximum flow in G*
2. *The residual network G_f contains no augmenting paths*
3. *$|f| = c(S, T)$ for some cut (S, T) of G*

Proof. Prove 3 parts

- (1) $\to$ (2) Prove by contradiction. Assume f is a maximum flow in G but G_f has an augmenting path p with flow f_p. If we augment f by f_p, we have $|f \uparrow f_p| = |f| + |f_p| > |f|$, implies there is a larger flow value, contradicting f is the maximum flow

- (2) $\to$ (3) Idea is to identify cut (S, T), infer value of $f(u, v)$ from the fact there exists no path from s to t in G_f, then calculate net flow $f(S, T)$ across an arbitrary cut, which is identical for any cut, including $|f|$. Assume G_f has no augmenting path, that is there is no path from s to t, Define

  $$S = \{v \in V : \text{ there exists a path from } s \text{ to } v \text{ in } G_f\}$$

  and $T = V \setminus S$. Consider vertices $u \in S$ and $v \in T$. $(u, v) \notin E_f$

  1. If $(u, v) \in E$, then $f(u, v) = c(u, v)$ since otherwise we have $c_f(u, v) = c(u, v) - f(u, v) > 0$, implying $(u, v) \in E_f$
  2. If $(v, u) \in E$, then $f(u, v) = 0$ since otherwise we have $c_f(u, v) = f(v, u) > 0$, implying $(u, v) \in E_f$
  3. If $(u, v) \notin E$ or $(v, u) \notin E$, hten $f(u, v) = f(v, u) = 0$

  Now we compute a net flow over the cut (S, T) in G

  $$f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{v \in T} \sum_{u \in S} f(v, u) = \sum_{u \in S} \sum_{v \in T} c(u, v) - \sum_{v \in T} \sum_{u \in S} 0 = c(S, T)$$

  By previous corollary, net flow is same for all arbitrary cuts, we have

  $$|f| = f(S, T) = c(S, T)$$

- (3) $\to$ (1) By previous corollary, the value of flow $|f|$ is bounded above by capacity of any cuts. $|f| \leq c(S, T)$. hence when $|f| = c(S, T)$ implies f is a maximum flow

$\square$

Definition. ***Ford-Fulkerson algorithm*** $O(E|f^*|)$

1. ***Steps***
   (a) *Initialize* $(u, v).f$ *to 0*
   (b) *Loop if there exists an augmenting path* p *from* s *to* t *in residual network* G_f
   (c) *Find residual capacity of the path* $c_f(p) = Min\{c_f(u, v) : (u, v) \text{ is in } p\}$ *in* G_f
   (d) *We replace* f *with* $f \uparrow f_p$ *to obtain a new flow whose value is* $|f| + |f_p|$
      i. *If* $(u, v) \in E$*, i.e. residual edge in* p *is an edge in the original network,* $(u, v) \in G_f$ *specifies how much flow* $(u, v) \in G$ *can increase by, so add* $c_f(p)$ *amount of flow to* $(u, v) \in G$
      ii. *If* $(v, u) \in E$*, i.e. residual edge in* p *is a reverse edge in the original network,* $(u, v) \in G_f$ *specifies how much flow* $(v, u) \in G$ *can decrease by, so decrease* $c_f(p)$ *amount of flow to* $(v, u) \in G$
2. ***Analysis*** *Runtime depends on finding the augmenting path* p*.*
   (a) ***Initialization*** $O(E)$
   (b) ***While loop*** *If capacity is rational, scale to integer. If* f^* *is the max flow,* FORD-FULKERSON *executes while loop at most* $|f^*|$ *times, since flow value increases by at least one unit in each iteration.*
   (c) ***Finding path*** *Assume we have a data structure representing a directed graph* $G' = (V, E')$ *where* $E' = \{(u, v) : (u, v) \in E \vee (v, u) \in E\}$*. The edges in* G_f *consists off all edges* $(u, v) \in E'$ *such that* $c_f(u, v) > 0$*. If use DFS, or BFS, runtime* $O(V + E') = O(E)$ *(since* $|E| \geq |V| - 1$*) for finding a path from* s *to* t*.*

   In summary runtime of $O(E|f^*|)$*. The algorithm is good if capacities are integral and the optimal flow value* $|f^*|$ *is small.*

26.3 Maximum Bipartite Matching $O(VE)$

Definition. ***Matching***

1. ***Matching*** *Given undirected graph* $G = (V, E)$*, a matching is a subset of edges* $M \subseteq E$ *such that for all vertices* $v \in V$*, at most one edge of* M *is incident on* v*. (each edge symbolizes a pair)*
2. ***Matched and Unmatched*** *a vertex* $v \in V$ *is matched by the matching* M *if some edge in* M *is incident on* v*; otherwise* v *is unmatched*
3. ***Maximum Matching*** *A maximum matching is a matching of maximum cardinality, that is, a matching* M *such that for any matching* M'*, we have* $|M| \geq |M'|$

4. ***Bipartite graphs*** *graphs in which V can be partitioned into 2 disjoint sets $V = L \cup R$, $L \cap R = \emptyset$ and all edges in E go between L and R. Assume every vertex in V has at least one incident edge*

Definition. ***Finding a Maximum Bipartite Matching***

1. ***Corresponding Flow Network*** *$G' = (V', E')$ (directed) for a bipartite graph $G = (V, E)$ (undirected) with partition $V = L \cup R$ is defined as follows*

   (a) *let source s and sink t be new vertices not in V*

   $$V' = V \cup \{s, t\}$$

   (b) *let directed edge of G' be edges of E, directed from L to R, along with $|V|$ new directed edges connecting s to L and R to t*

   $$E' = \{(s, u) : u \in L\} \cup \{(u, v) : (u, v) \in E\} \cup \{(v, t) : v \in R\}$$

   Note $|E'| = \Theta(E)$, since $|E| \geq |V|/2$ (every vertex has an incident edge) implies

   $$\Omega(E) = |E| \leq |E'| = |E| + |V| \leq 3|E| = O(E)$$

   (c) *assign unit capacity to each edge in E'*

2. ***Ingeter-valued flow*** *A flow f on a flow network G is integer valued if $f(u, v)$ is an integer for all $(u, v) \in V \times V$*

3. *A matching in G corresponds to a flow in G's corresponding flow network G'*

   Lemma. *Let $G = (V, E)$ be a bipartite graph with vertex partition $V = L \cup R$ and let $G' = (V', E')$ be corresponding flow network. If $M \subseteq E$ is a matching in G, then there is an integer-valued flow f in G' with value $|f| = |M|$. Conversely, if f is an integer-valued flow in G', then there is a matching M in G with cardinality $|M| = |f|$*

   Proof. 2 steps

   (a) Find matching M in G corresponds to flow f in G'. Define f as follows. If $(u, v) \in M$, then $f(s, u) = f(u, v) = f(v, t) = 1$. For all other edges $(u, v) \in E'$, define $f(u, v) = 0$. Hence each $(u, v) \in M$ corresponds to one unit of flow in G' traversing path

   $$s \to u \to v \to t$$

   The cut $(L \cup \{s\}, R \cup \{t\})$ is equal to $|M|$ by the previous definition, and hence $|f| = |M|$ (net flow same for any cuts)

(b) Prove converse. Let f be integer-valued flow in G', prove there is a matching such that $|M| = |f|$. Let

$$M = \{(u, v) : u \in L,\, v \in R,\, f(u, v) > 0\}$$

Prove M is a matching (i.e. all edge $v \in V$ has at most 1 edge $e \in M$ incident on v). For $u \in L$, has one entering edge (s, u) of one unit of flow, by flow conservation, must have one unit of flow leaving it. Since f integer-valued, one unit of flow enter on at most 1 edge and leave on at most 1 edge. Hence there cannot be 2 edges leaving $u \in L$ Hence, one unit of flow entering u if and only if there is exactly one vertex $v \in R$ such that $f(u, v) = 1$. Similar argument to R.

Hence maximum matching M in bipartite graph G corresponds to a maximum flow in its corresponding flow network G'. □

4. *By previous lemma, we can compute maximum matching in G by running max-flow algorithm on G', the following theorem guarantees the output from* FORD-FULKERSON *will be a integer-valued flow*

   Theorem. *If the capacity function c takes on only integral values, then maximum flow f produced by* FORD-FULKERSON *Method has the property that $|f|$ is an integer. Moreover, for all vertices u and v, the value $f(u, v)$ is an integer*

   Corollary. *The cardinality of a maximum matching M in a bipartite graph G equals the value of a maximum flow f in its corresponding flow network G'*

5. ***Steps***

   (a) *Create corresponding flow network G'*
   (b) *Run* FORD-FULKERSON
   (c) *Obtain maximum matching M from integer-valued maximum flow f found*

6. ***Runtime*** *Note any matching in bipartite graph has cardinality of*

$$|M| \leq Min(L, R) = O(V)$$

   the value of maximum flow in G' is hence $O(V)$, therefore maximum matching in a bipartite graph takes $O(|f^*|E') = O(VE') = O(VE)$, since $|E'| = \Theta(E)$