CSCE 629-601 Analysis of Algorithms

Fall 2022

Instructor: Dr. Jianer Chen
Office: PETR 428
Phone: (979) 845-4259
Email: chen@cse.tamu.edu
Office Hours: MWF 3:50pm–5:00pm

Teaching Assistant: Vaibhav Bajaj
Office: EABC 107B
Phone: (979) 739-2707
Email: vaibhavbajaj@tamu.edu
Office Hours: T; 2pm-3pm, TR: 4pm-5pm

Course Notes. Maximum Flow Algorithms

Suppose that we have built a network of pipes to transport oil in an area. Each pipe has a fixed capacity for pumping oil, measured by barrels per hour. Now suppose that we want to transport a very large quantity of oil from city s to city t. How do we use the network system of pipes so that the oil can be transported in the shortest time?

The problem can be modeled by the MAXIMUM FLOW problem. The network of pipes will be modeled by a directed graph G with two distinguished vertices s and t. Each edge in the graph G is associated with an integer, indicating the capacity of the corresponding pipe. Now the problem is to assign each edge with a flow, less than or equal to its capacity, so that the maximum amount of flow goes from vertex s to vertex t.

The MAXIMUM FLOW problem arises in many settings in operations research and other fields. In particular, it can be used to model liquids flowing through pipes, parts through assembly lines, current through electrical networks, information through communication networks, and so forth. Efficient algorithms for the problem have received a great deal of attention in the past decades.

1 Definitions and basic facts

We start with the formal definitions.

Definition A *flow network* $G = (V, E)$ is a directed graph with two distinguished vertices s (the *source*) and t (the *sink*). Each edge $[u, v]$ in G is associated with a positive integer $cap(u, v)$, called the *capacity* of the edge. If there is no edge from u to v, then $cap(u, v) = 0$.

Remark. There is no special restriction on the directed graph G that models a flow network. In particular, we allow edges in G to be directed into the source and out of the sink.

Intuitively, a flow in a flow network should satisfy the following three conditions:
(1) the amount of flow along an edge should not exceed the capacity of the edge (capacity constraint);
(2) a flow from a vertex u to a vertex v can be regarded as a "negative" flow of the same amount from vertex v to vertex u (skew symmetry); and
(3) except for the source s and the sink t, the amount of flow getting into a vertex v should be equal to the amount of flow coming out of the vertex (flow conservation).
These conditions are formally given in the following definition.

Definition A *flow* f in a flow network $G = (V, E)$ is an integer-valued function on pairs of vertices of G satisfying the following conditions:

1. For all $u, v \in V$, $cap(u, v) \geq f(u, v)$ (*capacity constraint*);
2. For all $u, v \in V$, $f(u, v) = -f(v, u)$ (*skew symmetry*);
3. For all $u \neq s, t$, $\sum_{v \in V} f(u, v) = 0$ (*flow conservation*).

An edge $[u, v]$ is *saturated* by the flow f if $cap(u, v) = f(u, v)$. A path P in the flow network G is *saturated* by the flow f if at least one edge in P is saturated by f.

Note that even when there is no edge from a vertex u to a vertex v, the flow value $f(u, v)$ can still be non-zero. For example, suppose that there is no edge from u to v but there is an edge from v to u of capacity 10, and that the flow value $f(v, u)$ is equal to 8. Then by the skew symmetry property, the flow value $f(u, v)$ is equal to -8, which is not 0.

However, if there is neither edge from u to v and nor edge from v to u, then the flow value $f(u, v)$ must be 0. This is because from $cap(u, v) = cap(v, u) = 0$, by the capacity constraint property, we must have $f(u, v) \leq 0$ and $f(v, u) \leq 0$. By the skew symmetry property, $f(v, u) \leq 0$ implies $f(u, v) \geq 0$, which together with $f(u, v) \leq 0$ gives $f(u, v) = 0$.

Figure 1 is an example of a flow network G with a flow, where on each edge $e = [u, v]$, we label a pair of numbers as "a/b" to indicate that the capacity of the edge e is a and the flow from vertex u to vertex v is b.

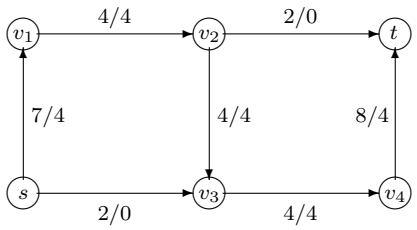

Figure 1: A flow network with a flow

Given a flow network $G = (V, E)$ with the source s and the sink t, let f be a flow on G. The *value* of the flow is defined to be $\sum_{v \in V} f(s, v)$, denoted by $|f|$.

Now the Maximum Flow problem can be formally defined as follow:

Maximum Flow:

Given a flow network (G, s, t), construct a flow in G with the largest value.

Our first observation on the properties of a flow is as follows.

Lemma 1 *Let $G = (V, E)$ be a flow network with the source s and the sink t, and let f be a flow in G. Then the value of the flow f is equal to $\sum_{v \in V} f(v, t)$.*

Proof. We have

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

By the skew symmetry property, $f(w,v) = -f(v,w)$. Note that in the sum $\sum_{w\in V}\sum_{v\in V} f(w,v)$, for each pair of vertices w and v, both $f(w,v)$ and $f(v,w)$ appear exactly once. Thus, we have $\sum_{w\in V}\sum_{v\in V} f(w,v) = 0$. Now apply the skew symmetry property on the second term on the right hand side, we obtain

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

Thus, we have

$$|f| = \sum_{w\neq s}\sum_{v\in V} f(v,w) = \sum_{w\notin\{s,t\}}\sum_{v\in V} f(v,w) + \sum_{v\in V} f(v,t).$$

Finally, by the skew symmetry and the flow conservation, for each $w \neq s,t$, we have

$$\sum_{v\in V} f(v,w) = -\sum_{v\in V} f(w,v) = 0.$$

Thus, the sum $\sum_{w\notin\{s,t\}}\sum_{v\in V} f(v,w)$ is equal to 0. This proves that $|f| = \sum_{v\in V} f(v,t)$. □

The following lemma describes a basic technique to construct a positive flow in a flow network.

Lemma 2 *Let $G = (V,E)$ be a flow network with the source s and the sink t. There is a flow f in G with a positive value if and only if there is a path in G from s to t.*

PROOF. Suppose that there is a path P from the source s to the sink t. Let e be an edge on P with the minimum capacity $c > 0$ among all edges in P. Now it is easy to see that if we assign flow of value c to each edge on the path P, and assign flow 0 to all other edges, we get a valid flow of value $c > 0$ in the flow network G.

For the other direction, suppose that f is a flow of positive value in the flow network G. Assume the contrary that there is no path in G from s to t. Let V' be the set of vertices in G that are reachable from s. Then $t \notin V'$.

Let w be a vertex in V'. We first show

$$\sum_{v\in V'} f(w,v) = \sum_{v\in V} f(w,v) - \sum_{v\notin V'} f(w,v) \geq \sum_{v\in V} f(w,v). \tag{1}$$

In fact, for any $v \notin V'$, since v is not reachable from the source s, there is no edge from w to v. Thus, $cap(w,v) = 0$, which implies $f(w,v) \leq 0$ by the capacity constraint property.

By Equation (1), we have

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

By the skew symmetry property, $\sum_{w\in V'}\sum_{v\in V'} f(w,v) = 0$. Thus,

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

For each $w \in V' - \{s\}$, according to Equation (1) and the flow conservation property, we have (note $t \notin V'$ so w cannot be t)

$$\sum_{v\in V'} f(w,v) \geq \sum_{v\in V} f(w,v) = 0.$$

Thus, we have $|f| \leq 0$. This contradicts our assumption that f is a flow of positive value. This contradiction shows that there must be a path in the flow network G from s to t. □

Thus, to construct a positive flow in a flow network G, we only need to find a path from the source to the sink. Many graph algorithms effectively find such a path.

2 Residual networks

One may suspect that finding a maximum flow is pretty straightforward: each time we find a path from the source to the sink, and add a new flow to saturate the path. After adding the new flow, if any edge becomes saturated, then it *seems* that the edge has become useless so we delete it from the flow network. For those edges that are not saturated yet, it seems reasonable to have a new capacity for each of them to indicate the amount of room left along that edge to allow further flow through. Thus, the new capacity should be equal to the difference of the original capacity minus the amount of flow through that edge. Now on the resulting flow network, we find another path to add further flow, and so forth.

One might expect that if we repeat the above process until the flow network contains no path from the source s to the sink t, then the obtained flow must be a maximum flow. Unfortunately, this observation is incorrect.

Consider the flow network with the flow in Figure 1. After deleting all saturated edges, the sink t is no longer reachable from the source s (see Figure 2). However, it seems that we still

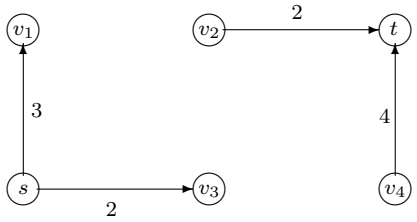

Figure 2: No s-t path exists after deleting saturated edges

can push a flow of value 2 along the "path" $s \to v_3 \to v_2 \to t$, where although we do not have an edge from v_3 to v_2, but we still can push a flow of 2 units from v_3 to v_2 by *reducing* the original flow by 2 units on edge $[v_2, v_3]$. This, in fact, does result in a larger flow in the original flow network, as shown in Figure 3.

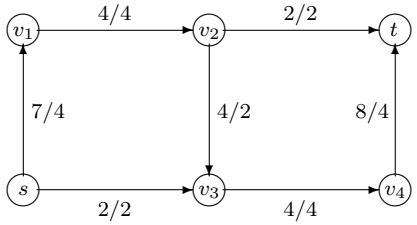

Figure 3: A flow larger than the one in Figure 1

Therefore, when a flow $f(u, v)$ is assigned on an edge $[u, v]$, it seems that not only do we need to modify the capacity of the edge $[u, v]$ to $cap(u, v) - f(u, v)$ to indicate the amount of further flow allowed through the edge, but also we need to record that a flow of amount $f(u, v)$ can be pushed along the opposite direction $[v, u]$, which is done by reducing the original flow along the edge $[u, v]$. In other words, we need add a new edge of capacity $f(u, v)$ from the vertex v to the vertex u. Motivated by this observation, we have the following definition.

Definition Given a flow network $G = (V, E)$ and a flow f in G, the *residual network* $G_f = (V, E')$ of G (with respect to the flow f) is a flow network that has the same vertex set V as G.

Moreover, for each ordered vertex pair (u, v), if $cap(u, v) > f(u, v)$, then $[u, v]$ is an edge in G_f with capacity $cap(u, v) - f(u, v)$.

Figure 4 is the residual network of the flow network in Figure 1 with respect to the flow given in the Figure. It can be clearly seen now that in the residual network, there is a path from s to t: $s \to v_3 \to v_2 \to t$.

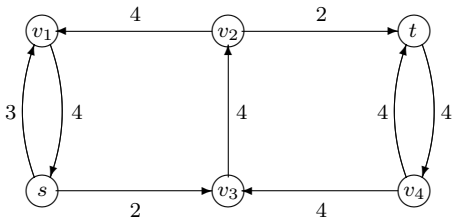

Figure 4: The residual network for Figure 1.

Remark. New edges may be created in the residual network G_f that were not present in the original flow network G. For example, there is no edge from vertex v_3 to vertex v_2 in the original flow network in Figure 1, but in the residual network in Figure 4, there is an edge from v_3 to v_2. However, if there is neither an edge from u to v nor an edge from v to u, then, since we must have $cap(u, v) = f(u, v) = 0$, there is also no edge from u to v in the residual network. This implies that the number of edges in a residual network cannot be more than twice of that in the original flow network. This fact will be useful when we analyze maximum flow algorithms.

Lemma 3 *Let G be a flow network and let f be a flow in G. If f^* is a flow in the residual network G_f, then the function $f^+ = f + f^*$, defined as $f^+(u, v) = f(u, v) + f^*(u, v)$ for all vertices u and v, is a flow with value $|f^+| = |f| + |f^*|$ in G.*

Proof. It suffices to verify that the function f^+ satisfies all the three constraints in the definition of a flow in a flow network. For each pair of vertices u and v in G, we denote by $cap(u, v)$ the capacity from u to v in the original flow network G, and by $cap_f(u, v)$ the capacity in the residual network G_f.

The Capacity Constraint. By the definition of the function f^+, we have

$$cap(u, v) - f^+(u, v) = cap(u, v) - f(u, v) - f^*(u, v).$$

By the definition of cap_f, $cap(u, v) - f(u, v) = cap_f(u, v)$. Moreover, since $f^*(u, v)$ is a flow in the residual network G_f, $cap_f(u, v) - f^*(u, v) \geq 0$. Consequently, we have $cap(u, v) - f^+(u, v) \geq 0$, i.e., $cap(u, v) \geq f^+(u, v)$.

The Skew Symmetry. Since $f(u, v)$ and $f^*(u, v)$ are flows in the flow networks G and G_f, respectively, we have $f(u, v) = -f(v, u)$ and $f^*(u, v) = -f^*(v, u)$. Thus,

$$f^+(u, v) = f(u, v) + f^*(u, v) = -f(v, u) - f^*(v, u) = -f^+(v, u).$$

The Flow Conservation. Again, since $f(u, v)$ and $f^*(u, v)$ are flows in the flow networks G and G_f, respectively, we have for all $u \neq s, t$

$$\sum_{v \in V} f^+(u, v) = \sum_{v \in V} f(u, v) + \sum_{v \in V} f^*(u, v) = 0.$$

This proves that f^+ is a flow in the flow network G. For the flow value of f^+, we have

$$|f^+| = \sum_{v \in V} f^+(s, v) = \sum_{v \in V} f(s, v) + \sum_{v \in V} f^*(s, v) = |f| + |f^*|.$$

This completes the proof of the lemma. □

Now we are ready for the following fundamental theorem for maximum flow algorithms.

Theorem 4 *Let G be a flow network and let f be a flow in G. The flow f is a maximum flow in G if and only if the residual network G_f has no positive flow, or equivalently, if and only if there is no path from the source s to the sink t in the residual network G_f.*

Proof. The equivalence of the second condition and the third condition is given by Lemma 2. Thus, we only need to prove that the first condition and the second condition are equivalent.

Suppose that f is a maximum flow in G. If the residual network G_f has a positive flow f^*, $|f^*| > 0$, then by Lemma 3, $f^+ = f + f^*$ is also a flow in G with flow value $|f| + |f^*|$. This contradicts the assumption that f is a maximum flow in G since $|f^*| > 0$. Thus, the residual network G_f has no positive flow.

For the other direction, we assume that f is not a maximum flow in G. Let $f_{\max}$ be a maximum flow in G. Thus, $|f_{\max}| - |f| > 0$. Now define a function f^- on each pair (u, v) of vertices in the residual network G_f as follows,

$$f^-(u, v) = f_{\max}(u, v) - f(u, v).$$

We claim that f^- is a valid flow in the residual network G_f.

The function f^- satisfies the capacity constraint: since $cap_f(u, v) = cap(u, v) - f(u, v)$,

$$cap_f(u, v) - f^-(u, v) = cap(u, v) - f(u, v) - f^-(u, v).$$

Note that $f(u, v) + f^-(u, v) = f_{\max}(u, v)$. Since $f_{\max}$ is a flow in G, $cap(u, v) - f_{\max}(u, v) \geq 0$. Consequently, we have $cap_f(u, v) - f^-(u, v) \geq 0$, i.e., $cap_f(u, v) \geq f^-(u, v)$.

The function f^- satisfies the skew symmetry condition:

$$f^-(u, v) = f_{\max}(u, v) - f(u, v) = -f_{\max}(v, u) + f(v, u) = -f^-(v, u).$$

The function f^- satisfies the flow conservation condition: for all $u \neq s, t$, we have

$$\sum_{v \in V} f^-(u, v) = \sum_{v \in V} f_{\max}(u, v) - \sum_{v \in V} f(u, v) = 0.$$

Thus, f^- is a valid flow in the residual network G_f. Moreover, since we have

$$|f^-| = \sum_{v \in V} f^-(s, v) = \sum_{v \in V} f_{max}(s, v) - \sum_{v \in V} f(s, v) = |f_{max}| - |f| > 0,$$

we conclude that the residual network G_f has a positive flow.

This completes the proof of the theorem. □

Algorithm. Ford-Fulkerson
Input: a flow network (G, s, t)
Output: a maximum flow f in G

1. **for** (each pair (u, v) of vertices in G) $f(u, v) = 0$;
2. construct the residual network G_f;
3. **while** (there is a positive flow in G_f) **do**
   construct a positive flow f^* in G_f;
   let $f = f + f^*$ be the new flow in G;
   construct the residual network G_f.

Figure 5: Ford-Fulkerson's method for maximum flow

Theorem 4 suggests a classical method (called *Ford-Fulkerson's method*), described in Figure 5 for constructing a maximum flow in a given flow network.

According to Lemma 2, there is a positive flow in the residual network G_f if and only if there is a directed path from the source s to the sink t in G_f. Such a directed path can be found by a number of efficient graph search algorithms. Thus, the condition in the **while** loop in step 3 in the algorithm **Ford-Fulkerson** can be easily checked. Theorem 4 guarantees that when the algorithm halts, the obtained flow is a maximum flow.

The only problem left is how we construct a positive flow in a given residual network G_f. In order to make the algorithm efficient, we need to adopt a strategy that constructs a positive flow in the given residual network G_f effectively so that the number of executions of the **while** loop in step 3 is as small as possible. Many algorithms have been proposed for finding such a positive flow. In the next section, we describe an important technique, the *shortest path saturation* method, for constructing a positive flow given a flow network. We will see that when this method is adopted, the algorithm **Ford-Fulkerson** is efficient.

3 Shortest path saturation method

The method of *shortest path saturation* is among the most successful methods in constructing a positive flow in the residual network G_f to limit the number of executions of the **while** loop in step 3 of the algorithm **Ford-Fulkerson**, where the *length* of a path is measured by the number of edges in the path, so a *shortest path* is a path with the minimum number of edges.

In the rest discussions, we will assume that the flow network G has n vertices and m edges.

We first briefly describe an algorithm suggested by Edmond and Karp. Edmond and Karp considered the method of constructing a positive flow for the residual network G_f by finding a shortest path from s to t in G_f and saturating it. Intuitively, each execution of this process saturates at least one edge from a shortest path from s to t in the residual network G_f. Thus, after $O(m)$ such executions, all shortest paths from s to t in G_f are saturated, and the distance from the source s to the sink t should be increased. This implies that after $O(nm)$ such executions, the distance from s to t should be larger than n, or equivalently, the sink t will become unreachable from the source s. Therefore, if we adopt Edmond and Karp's method to find positive flow in the residual network G_f, then the **while** loop in step 3 in the algorithm **Ford-Fulkerson** is executed at most $O(nm)$ times. Since a shortest path in the residual network G_f can be found in time $O(m)$ (using, for example, breadth first search), this concludes that Edmond-Karp's

algorithm finds the maximum flow in time $O(nm^2)$.

Dinic proposed a different approach. Instead of finding a single shortest path in the residual network G_f, Dinic finds *all* shortest paths from s to t in G_f, then saturates *all* of them. In the following, we give detailed description and analysis for Dinic's approach.

Definition Let (G, s, t) be a flow network. A flow f in G is a *shortest saturation flow* if
(1) $f(u, v) > 0$ implies that $[u, v]$ is an edge in a shortest path from s to t in G, and
(2) the flow f saturates every shortest path from s to t in G.

For each vertex v in a flow network G with source s and sink t, denote by $dist(v)$ the length of (i.e., the number of edges in) the shortest path in G from the source s to v (the *distance from s to v*). Similarly, if f is a flow in G, we let $dist_f(v)$ be the length of the shortest path from s to v in the residual network G_f.

Lemma 5 *Let G be a flow network with source s and sink t, and let f be a shortest saturation flow in G. Then $dist(t) < dist_f(t)$.*

Proof. First we note that if a vertex v is on a shortest path P from the source s to a vertex w in G, then the subpath of P from s to v is a shortest path from s to v. We first prove the following claim:

Claim. Let $[v, w]$ be an edge in the residual network G_f, then $dist(w) \leq dist(v) + 1$. Moreover, if $[v, w]$ is not an edge in the original flow network G, then $dist(w) < dist(v) + 1$.

Proof of the Claim. If $[v, w]$ is an edge in G, then since any shortest path from s to v plus the edge $[v, w]$ is a path from s to w, whose length cannot be smaller than $dist(w)$, we have $dist(w) \leq dist(v) + 1$.

If $[v, w]$ is not an edge in G, then $cap(v, w) = 0$. Since $[v, w]$ is an edge in the residual network G_f, by the definition of edges in G_f, we must have $0 = cap(v, w) > f(v, w)$. Thus, $f(w, v) > 0$. Since f is a shortest saturation flow, only for edges in shortest paths from s to t in G, f may have positive flow value. Thus $[w, v]$ must be an edge in a shortest path P from s to t in G. Then the subpath of P from s to w is a shortest path from s to w, and the subpath of P from s to v is a shortest path from s to v. This gives $dist(w) = dist(v) - 1 < dist(v) + 1$.

This completes the proof of the claim.

Now we are ready to prove the lemma. Let $P = (v_0, v_1, \ldots, v_{r-1}, v_r)$ be a shortest path in the residual network G_f from the source s to the sink t, where $v_0 = s$ and $v_r = t$. Thus, $dist_f(t) = r$, and all $[v_i, v_{i+1}]$, $0 \leq i \leq r - 1$, are edges in the residual network G_f. By the claim proved above, we have $dist(v_{i+1}) \leq dist(v_i) + 1$ for all i. Thus,

$$\begin{aligned} dist(t) &= dist(v_r) \\ &\leq dist(v_{r-1}) + 1 \leq dist(v_{r-2}) + 2 \leq \cdots \leq dist(v_0) + r \\ &= dist(s) + r \\ &= r = dist_f(t). \end{aligned}$$

This gives $dist(t) \leq dist_f(t)$. We prove by contradiction that $dist(t) < dist_f(t)$. Assume the contrary that $dist(t) = dist_f(t)$. Then none of the inequalities "$dist(v_{i+1}) \leq dist(v_i) + 1$" we used in the above derivation can be a strict inequality "$dist(v_{i+1}) < dist(v_i) + 1$"'. By the claim proved above, $[v_i, v_{i+1}]$ must be an edge in the original flow network G for all $0 \leq i \leq r - 1$.

But this would imply that the path $P = (v_0, v_1, \ldots, v_{r-1}, v_r)$ of length r is also a shortest path in the original flow network G (under the assumption $dist(t) = dist_f(t) = r$). However, since f is a shortest saturation flow in G, at least one $[v_i, v_{i+1}]$ of the edges in the path P should be saturated by f, i.e., $f(v_i, v_{i+1}) = cap(v_i, v_{i+1})$, but this would imply that $[v_i, v_{i+1}]$ is not an edge in the residual network G_f, contradicting the fact that P is a path in G_f. This contradiction proves that we must have $dist(t) < dist_f(t)$. □

Now we are ready to discuss how the shortest path saturation method is applied to the algorithm **Ford-Fulkerson**.

Theorem 6 *If in each execution of the* **while** *loop in step 3 in the algorithm* **Ford-Fulkerson**, *we construct a shortest saturation flow f^* for the residual network G_f, then the number of executions of the* **while** *loop is bounded by $n - 1$.*

PROOF. Suppose that f^* is a shortest saturation flow in the residual network G_f. By Lemma 5, the distance from s to t in the residual network $(G_f)_{f^*}$ (of G_f with respect to f^*) is at least 1 plus the distance from s to t in the original residual network G_f. Note that the residual network $(G_f)_{f^*}$ of G_f with respect to f^* is the residual network G_{f+f^*} of the original flow network G with respect to the new flow $f + f^*$. This can be easily verified by the following relation:

$$cap_f(u, v) - f^*(u, v) = cap(u, v) - (f(u, v) + f^*(u, v)) = cap(u, v) - [f + f^*](u, v).$$

Thus, $cap_f(u, v) > f^*(u, v)$ if and only if $cap(u, v) > [f + f^*](u, v)$, or equivalently, $[u, v]$ is an edge in $(G_f)_{f^*}$ if and only if it is an edge in G_{f+f^*}.

Therefore, the distance from s to t in the current residual network G_f is at least 1 plus the distance from s to t in the residual network G_f in the previous execution of the **while** loop. Since before the **while** loop, the distance from s to t in $G_f = G$ is at least 1 (the source s and the sink t are distinct in G), we conclude that after $n - 1$ executions of the **while** loop, the distance from s to t in the residual network G_f is at least n. This means that the sink t is not reachable from the source s in the residual network G_f. By Theorem 4, the algorithm **Ford-Fulkerson** stops with a maximum flow f. □

4 Dinic's algorithm

By Theorem 6, what is left is to construct a shortest saturation flow for the residual network G_f. By the definition, a shortest saturation flow saturates all shortest paths from s to t and has positive value only on edges on shortest paths from s to t. Thus, constructing a shortest saturation flow can be split into two steps: (1) finding all shortest paths from s to t in G_f, and (2) saturating all these paths.

Since there can be too many (up to 2^{cn} for some constant $c > 0$) shortest paths from s to t, it is infeasible to enumerate all of them. Instead, we construct a subnetwork L_0 in G_f, the *layered network*, that contains exactly those edges contained in the shortest paths from s to t.

The layered network L_0 of G_f can be constructed using a modified *breadth first search* process, as given in Figure 6, where Q is a queue that is a data structure serving for "first-in-first-out".

Stage 1 of the algorithm **Layered-Network** is a modification of the standard breadth first search process. The stage assigns a value $dist(v)$ to each vertex v, which equals the distance from the source s to v, and includes an edge $[v, w]$ in L_0 only if $dist(v) = dist(w) - 1$. The difference of this stage from the standard breadth first search is that for an edge $[v, w]$ with

Algorithm. Layered-Network
Input: the residual network $G_f = (V_f, E_f)$
Output: the layered network $L_0 = (V_0, E_0)$ of G_f

Stage 1. {constructing all shortest paths from s to each vertex}
1. $V_0 = \emptyset$; $E_0 = \emptyset$;
2. **for** all vertices v in G_f **do** $dist(v) = \infty$;
3. $dist(s) = 0$; $Q \leftarrow s$;
4. **while** ($dist(t) == \infty$ or t is in the queue Q) **do**
   $v \leftarrow Q$;
   for (each edge $[v, w]$)
   if ($dist(w) = \infty$)
   $Q \leftarrow w$; $dist(w) = dist(v) + 1$;
   $V_0 = V_0 \cup \{w\}$; $E_0 = E_0 \cup \{[v, w]\}$;
   else if ($dist(w) = dist(v) + 1$) $E_0 = E_0 \cup \{[v, w]\}$;

Stage 2. {deleting vertices not in a shortest path from s to t}
5. let L_0^r be L_0 with all edge directions reversed;
6. perform a breadth first search on L_0^r, starting from t;
7. delete the vertices v from L_0 if v is not marked in step 6.

Figure 6: Construction of the layered network L_0

$dist(v) = dist(w) - 1$, even if the vertex w has been in the queue Q, we still include the edge $[v, w]$ in L_0 to record the shortest paths from s to w that contain the edge $[v, w]$. Therefore, after stage 1, for each vertex v, exactly those edges contained in shortest paths from s to v are included in the network $L_0 = (V_0, E_0)$.

Stage 2 of the algorithm is to delete from L_0 all vertices (and their incident edges) that are not in shortest paths from the source s to the sink t. Since L_0 contains only shortest paths from s to each vertex and every vertex in L_0 is reachable from s in L_0, a vertex v is not contained in any shortest path from s to t if and only if t is not reachable from v in the network L_0, or equivalently, v is not reachable from t in the reversed network L_0^r. Step 6 in the algorithm identifies those vertices that are reachable from t in L_0^r, and step 7 deletes those vertices that are not identified in step 6.

Therefore, the algorithm **Layered-Network** correctly constructs the layered network L_0 of the residual network G_f. By the well-known analysis for the breadth first search process, the running time of the algorithm **Layered-Network** is bounded by $O(m)$.

Given the layered network L_0, Dinic's algorithm for saturating all shortest paths from s to t in L_0 (thus in G_f) is very simple, and can be described as follows. Starting from the vertex s, we follow the edges in L_0 to find a maximal path P of length at most $dist(t)$. Since the network L_0 is layered and contains only edges in the shortest paths from s to t in G_f, the path P can be found in a straightforward way (i.e., at each vertex, simply follow an arbitrary out-going edge from the vertex). Thus, the path P can be constructed in time $O(dist(t)) = O(n)$. Now if the ending vertex is t, then we have found a path from s to t. We trace back the path P to find the edge e on P with minimum capacity c. Now we can push c amount of flow along the path P. Then we delete the edges on P that are saturated by the new flow. Note that this deletes at least one edge from the layered network L_0. On the other hand, if the ending vertex v of P is not t, then v must be a "deadend". Thus, we can delete the vertex v (and all incoming edges to v). In conclusion, in the above process of time $O(n)$, at least one edge is removed from the layered

network L_0. Thus, after at most m such processes, the vertices s and t are disconnected, i.e., all shortest paths from s to t are saturated. This totally takes time $O(nm)$. A formal description for this process is given in Figure 7.

Algorithm. Dinic-Saturation
Input: the layered network L_0
Output: a shortest saturation flow f^* in G_f

1. **while** (there is an edge from s in L_0) **do**
   find a path P of maximal length from s in L_0;
   if (P leads to t) saturate P and remove the saturated edges;
   else delete the last vertex of P from L_0.

Figure 7: Dinic's algorithm for a shortest saturation flow

For completeness, we present in Figure 8 the complete Dinic's algorithm for constructing a maximum flow in a given flow network.

Algorithm. MaxFlow-Dinic
Input: a flow network (G, s, t)
Output: a maximum flow in (G, s, t)

1. **for** (each pair (u, v) of vertices in G) $f(u, v) = 0$;
2. construct the residual network G_f;
3. **while** (there is a positive flow in G_f) **do**
   call **Layered-Network** to construct the layered network L_0 for G_f;
   call **Dinic-Saturation** on L_0 to construct a shortest saturation flow f^* in G_f;
   $f = f + f^*$; and construct the residual network G_f;
4. return(f).

Figure 8: Dinic's algorithm for maximum flow

Theorem 7 *Dinic's maximum flow algorithm runs in time $O(n^2m)$.*

PROOF. Theorem 6 claims that the **while** loop in step 3 in the algorithm is executed at most $n - 1$ times. In each execution of the loop, constructing the layered network L_0 by **Layered-Network** takes time $O(m)$. Constructing the shortest saturation flow f^* in G_f from the layered network L_0 by **Dinic-Saturation** takes time $O(nm)$. All other steps in the loop takes time at most $O(n^2)$. Therefore, the total running time for the algorithm **MaxFlow-Dinic** is bounded by $O(n^2m)$. □