Problems: Flow Basics, Ford-Fulkerson Method, Edmond-Kard Algorithm

Palash Dey
Indian Institute of Technology, Kharagpur

1. [Simplifying Input Instance] Explain how we can assume without loss of generality that the input directed graph for the network flow problem does not contain any self-loops or anti-parallel edges. For two vertices u and v, if the graph has an edge from u to v and another edge from v to u, then these pair of edges are called anti-parallel edges.
2. [Fattest Augmenting Path] Run the fattest path first (augment along the path where the maximum amount of flow can be sent from s to t) implementation of the Ford-Fulkerson method and the Edmond-Karp algorithm on large random graphs and evaluate their relative performance by varying the number of vertices, edges, and average capacities of the edges. Use your favorite programming language.
3. Modify your above code to compute a minimum capacity $s-t$ cut in a flow network.
4. [CLRS book] Suppose that, in addition to edge capacities, a flow network has vertex capacities. That is each vertex v has a limit $\ell(v)$ on how much flow can pass though v. Show how to transform a flow network $\mathcal{G}(\mathcal{V}, \mathcal{E}, w : \mathcal{V} \longrightarrow \mathbb{R}_{>0})$ with vertex capacities into an equivalent flow network $\mathcal{G}'(\mathcal{V}', \mathcal{E}', w' : \mathcal{E} \longrightarrow \mathbb{R}_{>0})$ without vertex capacities, such that a maximum flow in $\mathcal{G}'$ has the same value as a maximum flow in $\mathcal{G}$. How many vertices and edges does $\mathcal{G}'$ have?
5. [Multiple Source and Multiple Sink] Suppose that we have multiple source and sink vertices in a flow network. The value of a flow in such a network is defined as the total amount of flow going out of all sources. Show how to transform this problem into an equivalent conventional (taught in the class) flow network such that the maximum flow value remains the same.
6. [Flow with Vertex Capacities] Suppose every vertex v, except s, t, has a capacity c_v, meaning that at most c_v units of flow may pass through v. Design an algorithm that computes a maximum $s-t$ flow subject to both edge capacities and vertex capacities.
7. [Maximum Flow with Lower Bounds] Each edge e has a lower bound l_e and upper bound u_e, and every feasible flow must satisfy $l_e \leqslant f_e \leqslant u_e$. Design an algorithm to determine whether a feasible $s-t$ flow exists. Then extend your algorithm to compute a maximum feasible s-t flow.
8. [Minimum Cut After Maximum Flow] Suppose a maximum flow has already been computed. Design an $O(|V|+|E|)$-time algorithm that finds an $s-t$ minimum cut.
9. [CLRS book] Suppose that you wish to find, among all minimum cuts in a flow network $\mathcal{G}$ with integral capacities, one that contains the smallest number of edges. Show how to modify the capacities of $\mathcal{G}$ to create a new flow network $\mathcal{G}'$ in which any minimum cut in $\mathcal{G}'$ is a minimum cut with the smallest number of edges in $\mathcal{G}$.
10. [Uniqueness of Maximum Flow] Design an algorithm to check if a flow network has a unique minimum capacity $s-t$ cut, and if not, then output two different minimum capacity $s-t$ cuts.
11. [Edge Capacity Increase] We are given a flow network and a maximum flow in that network. Assume all edge capacities are positive integers. Suppose we increase the capacity of an edge by 1. Give an $\mathcal{O}(m+n)$ time algorithm to update the maximum flow.
12. [Submodularity of Cuts] Let $\mathcal{G}(\mathcal{V}, \mathcal{E})$ be an undirected and unweighted graph. For $\mathcal{X} \subseteq \mathcal{V}$, we define $\delta(\mathcal{X})$ to be the capacity of the cut $(\mathcal{X}, \mathcal{V} \setminus \mathcal{X})$. Then show the following for every two subsets $\mathcal{A}, \mathcal{B} \subseteq \mathcal{V}$

$$\delta(\mathcal{A}) + \delta(\mathcal{B}) \geqslant \delta(\mathcal{A} \cup \mathcal{B}) + \delta(\mathcal{A} \cap \mathcal{B})$$

13. [Edge in Every Min Cut] Given a flow network, design an algorithm of worst-case time complexity $O(\text{time complexity of maximum } s-t \text{ flow computation})$ that identifies all edges that belong to every minimum $s-t$ cut.

14. [Flow with a Mandatory Edge] Given a network and a particular edge $e = (u, v)$, design an algorithm that determines whether there exists a maximum $s-t$ flow that sends strictly positive flow through e.

15. [Path decomposition of a flow] A flow f in a flow network $\mathcal{G}$ is called *not acyclic* if there exists a directed cycle in $\mathcal{G}$ such that each of its edges carries a positive amount of flow. Otherwise, f is called acyclic.

    (a) Prove that, for every flow f, there exists an acyclic flow f' of value the same as the value of f.

    (b) A *path flow* is a flow which assigns a positive flow only to the edges of an $s-t$ path. Prove that, every acyclic flow f can be written as the sum of at most m path flows.

    (c) Is the Ford-Fulkerson algorithm guaranteed to find an acyclic flow?

    (d) A *cycle flow* is a flow which assigns a positive flow only to the edges of a directed cycle. Prove that, every flow f can be written as the sum of at most m path flows and cycle flows. Given a flow network and a flow f, design an algorithm to find the above decomposition in $\mathcal{O}(mn)$ time.

16. [Dinic's Algorithm] A *blocking flow* f in a flow network $\mathcal{G}(\mathcal{V}, \mathcal{E}, c : \mathcal{E} \longrightarrow \mathbb{R}_{>0})$ is a flow such that, in every $s-t$ path, there is at least one edge e such that $c_e = f_e$.

    (a) Given a network $\mathcal{G}$, and an $s-t$ flow f, construct the BFS tree starting from s. Delete backward and cross edges of the BFS tree from the residual graph $\mathcal{G}_f$. We call the resulting graph the layered graph $\mathcal{L}_f$ with respect to f. Prove that $\mathcal{L}_f$ is acyclic.

    (b) Design an $\mathcal{O}(mn)$-time algorithm to compute a blocking flow f in $\mathcal{L}_f$. *Hint: Modify the standard DFS appropriately!*

    (c) Consider the following algorithm for computing a maximum flow in a network. Start with the zero flow. Compute a blocking flow in the corresponding layered graph', augment it with the current flow, and iterate until there is no path from s to t in the residual graph. Prove that the run time of this algorithm $\mathcal{O}(mn^2)$. *Hint: How does the distance (number of edges in a shortest path) of any vertex from s changes after every iteration?*