COL351: Analysis and Design of Algorithms **Fall 2024**

Tutorial Sheet 11

Announced on: Oct 20 (Sun)

- This tutorial sheet contains problems in network flow. When presenting your solutions, please draw/describe the flow network clearly. The correctness and running time arguments, if required, can be brief (1-2 sentences).
- The tutorial quiz problem for different groups will be selected from the following sets:
  - Group 3 (Oct 24, Thurs): Problems 1-3
  - Group 1 (Oct 28, Mon): Problems 1-5
  - Group 2 (Oct 29, Tue): Problems 1-6
  - Group 4 (Nov 01, Fri): Problems 1-7

Note: For some of the problems, we've only provided the main ideas and not the full solutions. We expect you to fill the gaps by yourself. If there are any doubts, you can contact your tutorial TA, head TA or the instructor.

---

1. In class, we discussed the maximum flow problem in directed graphs. In the *undirected* version of the problem, the input is an undirected graph $G = (V, E)$, a source vertex $s \in V$, a sink vertex $t \in V$, and an integer capacity $u_e \geqslant 0$ for each edge $e \in E$. Flows are defined exactly as before, and remain *directed*. Formally, a flow consists of two nonnegative numbers $f_{u,v}$ and $f_{v,u}$ for each undirected edge $\{u, v\} \in E$, indicating the amount of traffic traversing the edge in each direction. Conservation constraints (flow in = flow out) are defined as before. Capacity constraints now state that, for every undirected edge $e = \{u, v\} \in E$, the total amount of flow $f_{u,v} + f_{v,u}$ on the edge is at most the edge's capacity u_e. The value of a flow is the net amount going out of the source, i.e.,

$$\sum_{v:\{s,v\}\in E} f_{s,v} - \sum_{v:\{s,v\}\in E} f_{v,s}.$$

   Prove that the maximum flow problem in undirected graphs reduces to the maximum flow problem in directed graphs. That is, given an instance of the undirected problem, show how to (i) produce an instance of the directed problem such that (ii) given a maximum flow to this directed instance, you can recover a maximum flow of the original undirected instance. Your implementations of steps (i) and (ii) should run in linear time. Include a brief proof of correctness.

In the given undirected graph $G = (V, E)$, we want to construct a new directed graph G' by replacing each undirected edge $e = \{u, v\} \in E$ with two directed edges (u, v) and (v, u), each having capacity u_e, the same as the capacity of the original undirected edge. We claim that the maximum flow in this directed graph G' is equal to the maximum flow in the original undirected graph G.
Let $f'_{u,v}$ represent the flow on the directed edge (u, v) in the maximum flow for G'. We can then recover a maximum flow f for the undirected edges in G as follows:

$$f_{u,v} = \max(f'_{u,v} - f'_{v,u},\ f'_{v,u} - f'_{u,v},\ 0)$$

This construction ensures that at least one of $f_{u,v}$ or $f_{v,u}$ is zero, meaning that flow occurs in at most one direction along each undirected edge in G.
Claim: For any undirected edge $\{u, v\} \in E$, the total flow on that edge satisfies:

$$f_{u,v} + f_{v,u} \leqslant c_e$$

Proof: Without loss of generality, assume $f'_{u,v} \geqslant f'_{v,u}$. Then, based on our construction:

$$f_{u,v} = f'(u, v) - f'(v, u)$$

$$f_{v,u} = 0$$

Since $0 \leqslant f'_{u,v} \leqslant u_e$ and $0 \leqslant f'_{v,u} \leqslant u_e$, it follows that:

$$f_{u,v} = f'_{u,v} - f'_{v,u} \leqslant u_e$$

The case where $f'(v, u) \geqslant f'(u, v)$ is similar and leads to the same conclusion.

2. Suppose we generalize the maximum flow problem so that there are *multiple* source vertices $s_1, \ldots, s_k \in V$ and sink vertices $t_1, \ldots, t_\ell \in V$. (As usual, the rest of the input is a directed graph with integer edge capacities.) You should assume that no vertex is both a source and sink, that source vertices have no incoming edges, and that sink vertices have no outgoing edges. A flow is defined as before: a nonnegative number f_e for each edge $e \in E$ such that capacity constraints are obeyed on every edge and such that conservation constraints hold at all vertices that are neither a source nor a sink. The value of a flow is the total amount of outgoing flow at the sources, i.e.,

$$\sum_{i=1}^{k} \sum_{e \in \delta^+(s_i)} f_e$$

where $\delta^+(s_i)$ denotes the outgoing edges from the vertex s_i.

Prove that the maximum flow problem in graphs with multiple sources and sinks *reduces* to the single-source single-sink version of the problem. That is, given an instance of the multi-source multi-sink version of the problem, show how to (i) produce a single-source single-sink instance such that (ii) given a maximum flow to this single-source single-sink instance, you can recover a maximum flow of the original multi-source multi-sink instance.

Your implementations of steps (i) and (ii) should run in linear time. Include a brief proof of correctness.

Reduction: Create a dummy source node s' that has outgoing edges to all the other sources of our graph. Similarly, create a dummy sink node t' that has incoming edges from all the other sinks of our graph. Let refer to this new graph as $G^{'}$.
The capacity of the edge $(s', s_k) \forall k$ should be equal to the sum of capacity of all the edges leading out of s_k. Likewise, the capacity of the edge $(t_l, t') \forall l$ should be equal to the sum of capacity of all the edges coming into the t_l.
Now, any max-flow algorithm can be used on $G^{'}$ to find the max-flow in G.

Proof of Correctness: For a flow f, we denote its value by $|f|$.
Claim 1. *If there exists a flow f in G, then there exists a flow of value $|f|$ in $G^{'}$.*

Proof. We construct the flow $f^{'}$ for $G^{'}$ as follows: For all edges $e \in G^{'}$ such that e is neither an outgoing edge of s nor an incoming edge to t, let $f^{'}(e) = f(e)$. Also, for any outgoing edge (s, s_i) of s, let $f^{'}(e) = \sum_u (s_i, u) \in E$. Similarly, once can define the flow in the incoming edges to t. It can be seen that $f^{'}$ is a valid flow in $G^{'}$ and $|f^{'}| = |f^{'}|$. Hence, proved. □

Claim 2. *If there exists a flow $f^{'}$ in $G^{'}$, then there exists a flow of value $|f^{'|}$ in G.*

Proof. Similar to previous proof. □

From the above two claims, we can infer the following:
Lemma 1. *Max-flow of the graph $G^{'}$ is equal to the max-flow of the graph G.*

Time complexity: The new nodes $s^{'}$ and $t^{'}$ can be added into the graph in $O(k)$ and $O(l)$ time respectively. Max-flow of the original graph can be recovered in $O(1)$ time using the max-flow of the newly generated graph (since they are equal).

- Written "I do not know how to approach this problem" - 0.6 points
-
  - Correct constructing of $G^{'}$ from G - 1.5 points
  - Brief justification of why the max-flow in $G^{'}$ is equal to the max-flow in G - 1 point
  - Brief justification of linear time complexity - 0.5 points

3. For every positive integer U, show that there is an instance of the maximum flow problem with edge capacities in $\{1, 2, \cdots, U\}$ and a choice of augmenting paths so that the Ford-Fulkerson algorithm runs for at least U iterations before terminating. The number of vertices and edges in your networks should be bounded above by a constant independent of U. (This

shows that the algorithm is only "pseudopolynomial.")

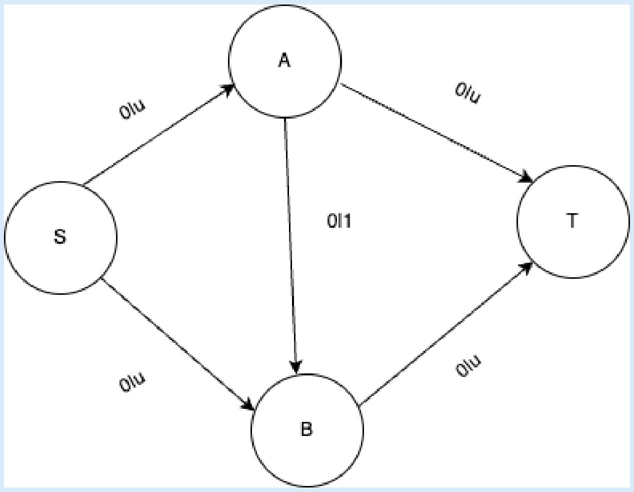

Figure 1: Example graph. For any edge, a/b denotes that a flow of a is passing through that edge currently, and b is the total capacity of that edge.

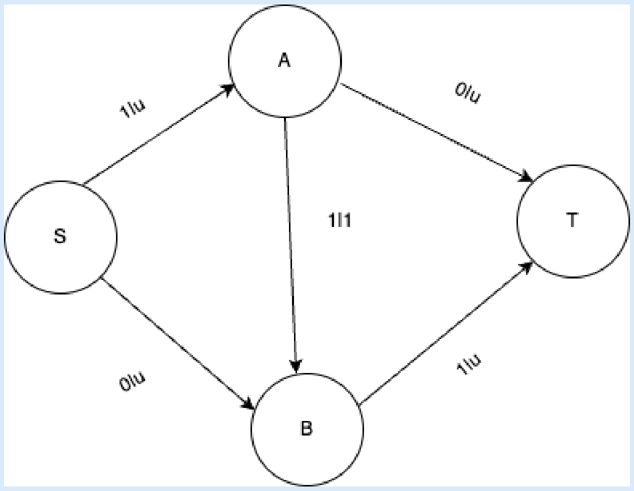

Figure 2: Resulting graph after choosing augmenting path $S \rightarrow A \rightarrow B \rightarrow T$

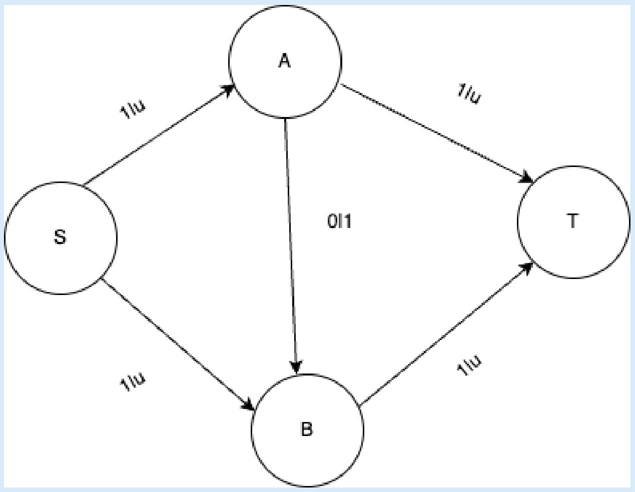

Figure 3: Resulting graph after choosing augmenting path $S \rightarrow B \rightarrow A \rightarrow T$

Consider the example shown in Figure ??. Suppose that in the first iteration of the Ford-Fulkerson algorithm, we select the augmenting path $S \rightarrow A \rightarrow B \rightarrow T$. The resulting flow in the graph after this iteration is shown in Figure ??.
In the second iteration, we choose a different augmenting path, $S \rightarrow B \rightarrow A \rightarrow T$. The resulting flow in the graph after this iteration is shown in Figure ??. Notably, the maximum flow in this graph will be $2U$.

If we continue alternating between these types of augmenting paths for the next $2U - 2$ iterations, we will reach the required maximum flow. This example demonstrates that the Ford-Fulkerson algorithm can be pseudopolynomial in the sum of the capacities of the edges leaving the source node.

4. Show that any $s - t$ flow can be written as a sum of flows along at most m $s - t$ paths and cycles, where m is the number of edges in the graph.

In an $s - t$ flow f, consider only the edges which have positive flow – let this graph be H. Consider any cycle C in H. It can be easily shown that one can remove some amount of flow from all edges of C so that at least one edge of H now carries 0 flow. We remove this edge(s) from H. Similarly, for any $s - t$ path in H, one can remove some amount of flow along this path so that at least one edge now carries 0 flow. So, in each iteration, we get a cycle or a path, and at least one edge is removed in every iteration. So, we get at most m cycles and $s - t$ paths.

5. Suppose in a directed graph G, there are k edge-disjoint paths from s to t and from t to u. Prove that there are at least k edge-disjoint paths from s to u.

Let $G = (V, E)$ and $s \in V, t \in V$ and $u \in V$
Using max-flow min-cut theorem: Let $X \subset V$ be a min-cut such that $s \in X$ and $u \notin X$.

a) $t \notin X$: X is an $s - t$ cut. Since, there are k edge-disjoint paths from s to t, the capacity of $s - t$ min-cut must be at least k (as all the edges that are part of this cut have the minimum edge capacity of 1).

b) $t \in X$: X is a $t - u$ cut. Since, there are k edge-disjoint paths from t to u, the capacity of $t - u$ min-cut must be at least k (as all the edges that are part of this cut have the minimum edge capacity of 1).

In both the cases, the cut defined by X has at least k capacity. But X actually defines $s - u$ cut. Thus, there should be k edge-disjoint paths from s to u.

- Written "I do not know how to approach this problem" - 0.6 points
-
    - Mentioning that k edge disjoint paths from s to t implies that the size of s, t min-cut is at least k - 0.5 points
    - Handling the case when $t \notin X$ - 1 point
    - Handling the case when $tinX$ - 1 point
    - Inferring that s, u min-cut is of size at least k and hence, k edge disjoint paths from s to u - 0.5 points

6. In the $s - t$ directed *edge-disjoint* paths problem, the input is a directed graph $G = (V, E)$, a source vertex s, and a sink vertex t. The goal is to output a maximum-cardinality set of edge-disjoint $s - t$ paths $P_1, \cdots, P_k$. That is, P_i and P_j should share no edges for each $i \neq j$, and k should be as large as possible.

   Prove that this problem reduces to the maximum flow problem. That is, given an instance of the disjoint paths problem, show how to (i) produce an instance of the maximum flow problem such that (ii) given a maximum flow to this instance, you can compute an optimal solution to the disjoint paths instance. Your implementations of steps (i) and (ii) should run in linear and polynomial time, respectively. (Can you achieve linear time also for (ii)?) Include a brief proof of correctness.

We can construct a max-flow problem from a disjoint-path problem by adding a capacity of 1 on all edges in the input graph.

Claim: The max-flow of the new graph is the same as the maximum-cardinality set of edge-disjoint $s - t$ paths in G.

This is trivially proven if we consider the integral flow theorem, which states that there always exists an integral maximum flow if each edge in a network has integral capacities. The integral flow in this case means that the flow in each edge is either 0 or 1, and the set of edges with flow 1 would form the disjoint paths.

Construction of disjoint-paths: (Assuming that the max-flow given to us is integral, i.e., flow through any edge is either 0 or 1):
Let G' be the output graph of the max-flow algorithm, we can get the disjoint paths in the following steps:

a) Let $\mathcal{P}$ be the set of disjoint paths, initially empty

b) Choose an arbitrary $s - t$ path P in G' in which each edge has positive flow assigned to it

c) Add P to $\mathcal{P}$

d) Delete all edges of P from G'

e) If no $s - t$ path exists in G' return $\mathcal{P}$, otherwise go back to step b.

For this algorithm to run in linear time you can think about how to choose an arbitrary $s - t$ path P in G' in $|P|$ time.

7. How will your answer to Problem 6 change if you are required to compute *vertex-disjoint* paths instead of edge-disjoint paths?

We can modify the input graph G as follows:

- For every vertex $v \neq s, t$, construct two vertices v_1, v_2. Let s and t remain as-is.
- Create the edge $v_1 \to v_2$ for every vertex $v \neq s, t$
- For every edge $u \to v$ create the edge $u_2 \to v_1$
- For every edge $s \to v$ create the edge $s \to v_1$
- For every edge $u \to t$ create the edge $u_2 \to t$

Now when we run the edge-disjoint path algorithm on this graph we would get the vertex disjoint path for the original graph. This is because for any vertex $v \neq s, t$ we can only choose at most one incoming edge into v_1 and at most one outgoing edge out of v_2, as there is only one edge between v_1 and v_2. Hence the edge-disjoint path would also be vertex disjoint.

8. ($\star$) Suppose we are given a $m \times n$ matrix A of nonnegative real numbers. We want to *round* A to an integer matrix, by replacing each entry x in A with either $\lceil x \rceil$ or $\lfloor x \rfloor$, without changing the sum of entries in any row or column of A. (Assume that all row and column sums of A are integral.) For example:

$$\begin{bmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{bmatrix} \to \begin{bmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{bmatrix}.$$

   a) Prove that such a rounding is guaranteed to exist.

   b) Describe and analyze an efficient algorithm that rounds A in this fashion.

a) To simplify the proof we will assume that each entry A_{ij} is between 0 and 1. Observe that for any matrix A, we can define A' by letting $A'_{ij} = A_{ij} - \lfloor A_{ij} \rfloor$. If we find a rounding B' of A', we can construct B by setting $B_{ij} = B'_{ij} + \lfloor A_{ij} \rfloor$. It's easy to verify that B is a rounding of A.

we construct a graph with nodes R_i representing each row and nodes C_j representing each column, and edges directed from R_i to C_j. In addition, we add two more nodes, s and t, into the graph. For each R_i, we add an edge from s to R_i and for each C_j, we add an edge from C_j to t.

Now we convert this graph into a flow graph. To do so, we specify an edge capacity u_e for every edge e. The rules are as follows:

i. $\forall i = 1, \ldots, n, j = 1, \ldots, n,\ u(R_i, C_j) = 1$

ii. $\forall i = 1, \ldots, n,\ u(s, R_i) = \sum_{j=1}^{m} A_{ij}$

iii. $\forall i = 1, \ldots, m,\ u(C_i, t) = \sum_{j=1}^{n} A_{ji}$

Note that rules 2 and 3 essentially mean that the capacity of (s, R_i) is the i-th row sum of A and the capacity of (C_i, t) is the i-th column sum of A. As the row and column sum are integers, the edge capacities are all integers.

For this flow graph, the value of a max-flow from s to t is at most $\sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij}$, as that's the maximum amount of flow going out of s or into t. Furthermore, we know that this value can be achieved — we just need to set the flow of (R_i, C_j) to be A_{ij} and saturate the edges from s to R_i and C_j to t. We can easily verify that such a flow satisfies both capacity and conservation constraints and has a value equal to $\sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij}$. Hence $\sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij}$ is indeed the value of a max-flow from s to t for this flow graph.

But remember, the max-flow is not unique — in fact, let the flow we get by running Edmonds-Karp(EK) be f. This flow f has the following properties:

i. $\forall e \in E,\ f_e \in \mathbb{Z}$.

ii. $\forall i = 1, \ldots, n,\ f(s, R_i) = u(s, R_i)$ and $f(C_i, t) = u(C_i, t)$. In other words, all edges going out of s or into t must be saturated.

Property 1 holds because for each augmenting step of EK , the augmenting path we find must have integer capacity for each edge included, thus the increment in the amount of flow for each edge must also be integral. Therefore, f is an integer flow.

Property 2 holds because we already know that the max-flow value is $\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}$, which means that the amount of flow going out of s or going into t must be equal to $\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}$. This can only be achieved when all edges going out of s or into t are saturated.

Now, define matrix B by setting $B_{ij} = f(R_i, C_j)$. Since f is an integer flow, each entry of B must also be an integer. Furthermore, combining property 2 above and conservation constraints we have:

$$\forall i = 1, \ldots, n, \sum_{j=1}^{n} f(R_i, C_j) = f(s, R_i) = u(s, R_i) = \sum_{j=1}^{n} A_{ij}$$

$$\forall i = 1, \ldots, n, \sum_{j=1}^{n} f(R_j, C_i) = f(C_i, t) = u(C_i, t) = \sum_{j=1}^{n} A_{ji}$$

Hence, replacing $f(R_i, C_j)$ with B_{ij}, we see that each row and column sum of B is the same as the corresponding row and column sum of A. Therefore, B is a rounding of A.

b) The above proof provides a practical way to find a rounding of A:

i. Construct a flow graph of A.

ii. Run a max-flow algorithm with polynomial runtime (such as EK) on the constructed flow graph to get a max-flow f.

iii. Convert f into a new matrix B, which is a rounding of A.

Time Complexity: Their are total $O(m+n)$ nodes and $O(mn)$ edges in the constructed flow graph. Therefore the time complexity of constructing rounding of A using EK ($O(nm^2)$) will be $O(m^2n^2(m+n))$.

9. ($\star$) Consider the following modification to the Ford-Fulkerson algorithm: Among all possible $s-t$ paths in the residual graph, pick the one such that the maximum amount of flow can be sent along this path. How will you find such a path? Show that you will need at most $m \cdot \log(nU)$ iterations. What is the overall running time?

Hints:

a) Show how to modify Dijkstra's shortest-path algorithm, without affecting its asymptotic running time, so that it computes an $s-t$ path with the maximum-possible minimum residual edge capacity.

b) Suppose the current flow f has value F and the maximum flow value in G is F^*. Prove that there is an augmenting path in G_f such that every edge has residual capacity at least $(F^* - F)/m$, where $m = |E|$. You may find it helpful to refer to the paths-and-cycles decomposition of a flow (Problem 4).

c) You might find the inequality $(1-x) \leqslant e^{-x}$ for $x \in [0,1]$ useful.

Finding the $s-t$ path with the maximum-possible minimum residual edge capacity: Already discussed in one of the previous Tutorials (see Tutorial 5 Problem 3).

Bounding the number of iterations: Let the total number of iterations be k, with f_i denoting the flow sent by the algorithm in i^{th} iteration, assuming $f_0 = 0$. Let $F_i = \sum_{j=0}^{i} f_j$ and denote by G_i the residual graph at the end of i^{th} iteration.
Hence, $G = G_0$ and $s-t$ max-flow in G is equal to F_k. Also, $s-t$ max-flow in G_i is equal to $F_k - F_i$.

Consider any $s-t$ max-flow of G_i, then by Problem 4, it can be written as sum of flows along at most m paths and cycles. Since flow along a cycle can be cancelled without affecting the flow value, the $s-t$ max-flow value of G_i can be written as sum of values of flows sent across at most m paths. Since the total flow value over all these paths is equal to $F_k - F_i$, there exists at least one path P_i along which a flow of $(F_k - F)/m$ can be sent and hence, every edge has residual capacity of $(F_k - F_i)/m$ along this path P_i.
For f_{i+1}, since we select the path with maximum-possible minimum residual capacity in G_i (which is at least $(F_k - F_i)/m$ by above arguments), we get that:

$$f_{i+1} \geqslant \frac{F_k - F_i}{m}$$

Note that:

$$F_k - F_i = F_k - F_{i-1} - f_i \leqslant F_k - F_{i-1} - \frac{F_k - F_i}{m}$$

$$\implies F_k - F_i \leqslant (1 - \frac{1}{m})(F_k - F_{i-1})$$

Doing this repeatedly, we get that;

$$\implies F_k - F_i \leqslant (1 - \frac{1}{m})^i (F_k - F_0) = (1 - \frac{1}{m})^i F_k$$

Putting $i = k-1$, we get that:

$$F_k - F_{k-1} \leqslant (1 - \frac{1}{m})^{k-1} F_k$$

$$\implies f_k \leqslant (1 - \frac{1}{m})^{k-1} F_k$$

Since the capacities are integral, all f_i's are also integral and hence, f_k is integral. We also know that f_k is non-zero and hence, we can conclude that $f_k \geqslant 1$.

$$\implies (1 - \frac{1}{m})^{k-1} F_k \geqslant 1$$

$$\implies e^{-\frac{k-1}{m}} F_k \geqslant 1$$

$$\implies \log F_k \geqslant \frac{k-1}{m}$$

$$\implies k = O(m \log(F_k)) = O(m \log(nU))$$