Network Flow: Introductory Problems
Due: N.A.

Panos Karagiannis
ID: ——
Graduate Student UCSC
Contents

<table>
<thead>
<tr>
<th>Question</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question 1 : Doing Dishes</td>
<td>3</td>
</tr>
<tr>
<td>Question 2 : Fussy Eaters</td>
<td>5</td>
</tr>
<tr>
<td>Question 3 : Verbal Assault</td>
<td>8</td>
</tr>
<tr>
<td>Question 4 : Mi-k Drop</td>
<td>12</td>
</tr>
<tr>
<td>Question 5 : Problem 25 Chapter 7 Kleinberg-Tardos</td>
<td>13</td>
</tr>
</tbody>
</table>
Question 1 : Doing Dishes

Consider a set $S$ of $n$ computer scientist roommates who would like to determine “who will do the dishes after dinner” for the next $m$ nights. Not all roommates have dinner at home every night, so let $S_i$ denote the subset of roommates who will have dinner at home on the $i$-th night and let $w_i \leq |S_i|$ be the number of people needed to do the dishes on the $i$-th night (some nights multiple people are needed). For each roommate $j \in S$ and night $1 \leq i \leq m$, let $Q_i(j) = 0$ if $j$ does not eat at home on the $i$-th night, otherwise let $Q_i(j) = w_i/|S_i|$, i.e., if $j$ eats at home on the $i$-th night. Now, for each roommate $j$, let:

$$R_j = \sum_{i=1}^{m} Q_i(j)$$

Clearly, we can’t hope that everyone does dishes exactly $R_j$ nights since, in general, $R_j$ won’t even be an integer. Nevertheless, the computer scientists claim:

There exists a dish-washing plan under which each roommate $j \in S$ does the dishes at most $\lceil R_j \rceil$ times

Prove the claim.

Answer:

First we formulate this problem as a network flow problem:

- Let $p_j$ be the vertex corresponding to roommate $j$
- Let $q_i$ be the vertex corresponding to day $i$
- Draw a directed edge $e = (q_i, p_j)$ with capacity $w_i$, if on day $q_i$ person $p_j$ eats at home
- Let $s$ be the source. Draw the directed edges $e = (s, q_i)$ with capacity $w_i$, for all vertices $q_i$
- Let $t$ be the sink. Draw the directed edges $e = (p_j, t)$ with capacity $\lceil R_j \rceil$ for all vertices $p_j$

Schematically, this is shown below when we consider $m = 5$ nights and $n = 4$ roommates (edges $(q_i, p_j)$ have been chosen arbitrarily):
Claim 1. If a flow of value $\sum_{i}^{m} w_{i}$ exists, then there exists a schedule under which each roommate does the dishes at most $\lceil R_{j} \rceil$ times.

Proof. If a flow with value $\sum_{i}^{m} w_{i}$ exists then an integral flow of value $\sum_{i}^{m} w_{i}$ also exists. Therefore, all the edges $e = (s, q_{i}), \forall i \in \{1 \ldots m\}$ are saturated and have flow equal to $w_{i}$.

By the “Flow Conservation” theorem and the capacity of the edges connecting roommates to the sink $t$, no roommate $j$ will have to clean the dishes more than $\lceil R_{j} \rceil$ times. □

Claim 2. A flow of value $\sum_{i}^{m} w_{i}$ always exists

Proof. We can always create a fractional flow whose total value is $\sum_{i}^{m} w_{i}$ if:

- We saturate all edges $(s, q_{i})$.
- For every edge $e = (q_{i}, p_{j})$ we supply flow $Q_{i}(j)$.

Since $\deg(q_{i}) = |S_{i}|$, then for every night $q_{i}$ we can send flow of total value $w_{i}$ from $s$ to $t$, by sending $Q_{i}(j) = w_{i}/|S_{i}|$ units of flow across $s-t$ paths: $(s, q_{i}, p_{j_{1}}, t), (s, q_{i}, p_{j_{2}}, t), \ldots (s, q_{i}, p_{j_{|S_{i}|}}, t)$.

Further notice that since $w_{i} \leq |S_{i}|$ we always respect the capacity conditions of edges $(q_{i}, p_{j})$ since: $Q_{i}(j) = w_{i}/|S_{i}| \leq cap(q_{i}, p_{j}) = 1$.

Since there are $m$ nights in total, at every vertex $p_{j}$ the incoming value of the flow will be:

$$\sum_{i=1}^{m} Q_{i}(j) = \sum_{i=1}^{m} w_{i}/|S_{i}| = R_{j}$$

Since every edge $(p_{j}, t)$ has capacity $\lceil R_{j} \rceil \geq R_{j}$ the incoming flow at any vertex $p_{j}$ will never exceed the capacity of the edge $(p_{j}, t)$. Hence, the constructed flow is valid and has value $\sum_{i}^{m} w_{i}$. □

Question 2 : Fussy Eaters

You are planning a dinner party for $n$ friends who, naturally, have all sorts of dietary restrictions and allergies. Your plan is to cook some appetizer dishes and some main dishes and portion them so that you end up with $n$ appetizer portions and $n$ main portions. You email everyone, and for each person $i \in \{1, \ldots, n\}$ you get back their list $A_{i}$ of OK-to-eat appetizer dishes and their list $M_{i}$ of OK-to-eat main dishes. Design an efficient algorithm that takes as input the number of portions of each dish and the lists $A_{i}, M_{i}$, designs a max flow instance, has it solved, and uses the (value of the) maximum flow to decide whether or not it is possible to give each friend an appetizer and a main portion that is OK for them to eat.

Answer: In order to reduce the complexity of this problem we can simplify it by considering two separate problems. Namely, since each guest has to eat one dish as an appetizer and another dish as main course we can solve two problems independently:
(a) Assign one appetizer to the friends, given lists $A_i$ and the quantity of each appetizer

(b) Assign one main dish to the friends, given lists $M_i$ and the quantity of each main dish

So let $A$ be the concatenation of all the lists $A_i$, let $M$ be the concatenation of all the lists $M_i$, let $a$ be the list with the number of portions of each appetizer and let $b$ be the list with the number of portions of each main course.

Algorithm 1 Solution to Problem 2

1: function FindMatching($A, M, a, m$)
2:   (a) Create a network, using $A, a$, to decide if it is possible to assign one appetizer to
3:     the friends such that each friend gets one appetizer (network formulation is discussed below)
4:   (b) Create a network, using $M, m$ to decide if it is possible to assign one main course to
5:     the friends such that each friend gets one appetizer (network formulation is discussed below)
6:   If the answer to both (a), (b) is true, then we return that it is possible to give each
7:     friend an appetizer and a main portion that is OK for them to eat, otherwise, we return false
8: end function

First, notice that if we solve problem (a) then by complete analogy we can also solve problem (b). So, we first proceed by creating a network instance whose value of maximum flow solves problem (a):

- Let $g_i$ be the vertex that corresponds to friend $i$.
- Let $a_j$ be the vertex that corresponds to appetizer $j$.
- Draw a directed edge $e = (g_i, a_j)$ with capacity 1, if friend $i$ has included appetizer $j$ in his preference list $A_i$.
- Let $s$ be the source. Draw the directed edges $e = (s, g_i)$ with capacity 1, for all vertices $g_i$.
- Let $t$ be the sink and the portion of each appetizer be $p_j$. Draw the directed edges $e = (a_j, t)$ with capacity equal to $p_j$.

An example graph with the corresponding capacities of each edge, 4 friends and 8 appetizers is shown schematically (edges $(g_i, a_j)$ have been chosen arbitrarily):
Correctness:
Correctness is based on the next two claims:

**Claim 3.** If there exists flow $f$ that has value $v(f) = n$, then an assignment exists where each friend $g_i$ gets one appetizer $a_j$ that is OK for him to eat.

*Proof.* If a flow of value $n$ exists then by the “Integrality Theorem” an integral flow of value $n$ exists. If an integral flow of value $n$ exists, it means that all edges $(s, g_i)$ will be saturated (blue edges). Hence, by flow conservation, exactly one unit of flow will leave each vertex $g_i$, essentially forcing each friend to make an appetizer choice. Since all the edges $(a_j, t)$ have capacity $p_j$ it implies that no more than $p_j$ friends can choose the same appetizer (green edges). As a result, a possible assignment exists, where each friend chooses one appetizer to eat while respecting the available capacity of each dish.

**Claim 4.** Given an assignment where each friend $g_i$ gets one appetizer $a_j$ that is OK for him to eat, we can always construct a flow $f$ with value $v(f) = n$.

*Proof.* Given such a feasible assignment then we send one unit of flow from $s$ to $t$ along each of the paths $(s, g_i, a_j, t)$, where friend $i$ is connected to appetizer $j$. This is a valid flow since it does not violate the capacity conditions, in particular on the edges $(a_j, t)$ due to the availability constraints $p_j$.

In a similar fashion we compute the analogous problem with the main dishes. In order to decide if a possible assignment exists where each friend eats one main dish, we simply replace $A_i$ with $M_i$ and repeat the above steps.

---

Question 2: Fussy Eaters continued on next page...
Runtime Analysis

We need to solve two network flow problems in order to determine whether a possible assignment exists.

If we let the number of friends be $F$ and the number of appetizers be $A$ then, to solve the first network flow problem, we simply have to run a max-flow algorithm on a graph with $O(F + A)$ nodes and $O(FA)$ edges.

Similarly to solve the second network flow problem, if we let the number of main dishes be $M$ we simply have to run a max-flow algorithm on a graph with $O(F + M)$ nodes and $O(FM)$ edges.

Ultimately, the total running time depends on the algorithm employed to solve the max-flow problems.

Question 3 : Verbal Assault

You are an English teacher assigning presentations to students. Each of your $n$ students will receive a topic and 31 fancy words that they must use in discussing the topic. You’ve announced the set of topics $T$ and the set of fancy words $F$, and have received from each student $i \in \{1, ..., n\}$ a set $T_i \subseteq T$ of topics the student is interested in, and a set $F_i \subseteq F$ of fancy words that the student would like to use.

You want to see if it is possible to assign to each student a topic and exactly 31 words such that:

- Each topic is assigned to at most 2 students.
- Each fancy word $w_i$ is assigned to at most $t_i$ students.

Describe a method that takes as input the sets $T_i$ and $F_i$ and the numbers $t_1, ... t_k$ and returns either “No”, or an assignment of topics and words that meets the requirements.

Answer:

We can simplify the problem by making the observation that the words are independently chosen from the topics. As a result, we can solve two network flow instances and at the end return either a Yes or a No.
Algorithm 2 Solution to Problem 2

1: function FindMatching
2: 
3: Create network $G_1$ to decide if it is possible to assign one topic to every student such that
4: each topic is assigned to at most 2 students (network formulation is discussed below)
5: 
6: If yes store $s - t$ paths used by the maximum flow, otherwise return No
7: 
8: Create network $G_2$ to decide if it is possible to assign 31 fancy words to each student
9: such that each word is assigned to at most $t_i$ students (network formulation is discussed below)
10: 
11: If yes store $s - t$ paths used by the maximum flow, otherwise return No
12: 
13: Use the $s - t$ paths used by the maximum flows for graphs $G_1, G_2$ respectively,
14: to retrieve the information of the assignment of topics and words to students (explained in detail
15: after the network formulation)
16: end function

We first start by determining if a topic can be given to every student while respecting the
limitation that each topic is assigned to at most 2 students. So, we construct the following
graph:

- Let $g_i$ be the vertex that corresponds to student $i$.
- Let $a_j$ be the vertex that corresponds to topic $j$.
- Draw a directed edge $e = (g_i, a_j)$ if student $i$ is interested in topic $j$, with capacity 1.
- Let $s$ be the source. Draw the directed edges $e = (s, g_i)$ with capacity 1, for all
  vertices $g_i$.
- Let $t$ be the sink. Draw the directed edges $e = (a_j, t)$ with capacity equal to 2 for all
  vertices $a_j$.

An example graph along with the capacities of each edge, is represented schematically in the
following figure (edges $(g_i, a_j)$ have been chosen arbitrarily):
Correctness:
Correctness is based on the next two claims:

**Claim 5.** If a flow \( f \) exists with \( v(f) = n \) then, an assignment also exists where each student chooses one topic and each topic is assigned to at most 2 students.

**Proof.** From the “Integrality Theorem”, if a flow of value \( n \) exists, then an integral flow of value \( n \) also exists. Since this is an integral flow and the capacities of all edges \((s, g_i)\) are 1, we conclude that a flow of value 1 must reach each node \( g_i \). Then, by the conservation theorem, for each student \( i \) there must exist a topic \( j \) such that edge \((g_i, a_j)\) carries one unit of flow. So, we assign each student \( i \) to a topic \( j \) if the edge \((g_i, a_j)\) carries one unit of flow. Finally we observe that the capacity conditions for every edge \((a_j, t)\) ensure that no topic is assigned to more than two students. \( \square \)

**Claim 6.** If an assignment exists where each student chooses one topic and each topic is assigned to at most 2 students, then we can always construct a flow \( f \) of value \( v(f) = n \).

**Proof.** Given a feasible assignment, then we send one unit of flow from \( s \) to \( t \) along each of the paths \((s, g_i, a_j, t)\), where student \( i \) has been assigned topic \( j \). This is a valid flow since it does not violate the capacity conditions. In particular the edges \((a_j, t)\) have capacity 2 which can never be exceeded by the value of the flow at vertex \( a_j \) since each topic can be assigned to at most 2 students. Clearly, since there exist \( n \) students and each one chooses a topic, the value of the constructed flow is also \( n \). \( \square \)

Now we deal with the problem of assigning each fancy word \( w_i \) to at most \( t_i \) students. Again we translate this problem into a network flow instance:
- Let \( g_i \) be the vertex that corresponds to student \( i \).
- Let \( w_j \) be the vertex that corresponds to topic \( j \).
- Draw a directed edge \( e = (g_i, w_j) \) if student \( i \) is interested in using word \( j \), with capacity 1.
- Let \( s \) be the source. Draw the directed edges \( e = (s, g_i) \) with capacity 31, for all vertices \( g_i \).
- Let \( t \) be the sink. Draw the directed edges \( e = (w_j, t) \) with capacity equal to \( t_j \) for all vertices \( w_j \).

An example graph along with the capacities of each edge, is represented schematically in the following figure (edges \( (g_i, w_j) \) have been chosen arbitrarily):

Correctness:
Correctness is based on the next two claims:

Claim 7. If a flow \( f \) exists with \( v(f) = 31n \) then, there exists an assignment where each student chooses 31 words and each word \( w_j \) is assigned to at most \( t_j \) students.

Proof. From the “Integrality Theorem”, if a maximum flow of value \( 31n \) exists, then an integral flow of value \( 31n \) also exists. Hence, all edges \( (s, g_i) \) must have flow equal to 31, so we conclude that a flow of value 31 must reach each node \( g_i \). Then, by the conservation theorem, for each student \( i \) there must exist 31 words \( j \) such that edges \( (g_i, w_j) \) carry one unit of flow. So, we assign each student \( i \) to a word \( j \) if the edge \( (g_i, w_j) \) carries one unit of
flow. Finally we observe that the capacity conditions for every edge \((w_j, t)\) ensure that no
topic is assigned to more than \(t_j\) students.

**Claim 8.** If a valid assignment of words to students exists so that each student chooses 31
words and each word \(w_j\) is assigned to at most \(t_j\) students, then there also exists a flow \(f\)
such that \(v(f) = 31n\).

**Proof.** Given such an assignment, then we send 31 units of flow from \(s\) to \(t\) along each of the
paths \((s, g_i, w_{j1}, t)\) ... \((s, g_i, w_{j31}, t)\), where student \(i\) has been assigned 31 words \(j_k\). This is
a valid flow since it does not violate the capacity conditions. In particular the edges \((w_j, t)\)
have capacity \(t_j\) which can never be exceeded by the value of the flow at vertex \(w_j\) since each
word can be assigned to at most \(t_j\) students. Clearly, since there exist \(n\) students and each
one chooses 31 words, the value of the constructed flow is also 31\(n\).

**Runtime Analysis:**

We need to solve two network flow problems in order to determine whether a possible assign-
ment exists. Also, if a feasible assignment of topics and words exists then we can retrieve
the assignment by looking at the \(s-t\) paths used by our flow in constant time (Using Claims
5 and 7).

If we let the number of students be \(G\) and the number of topics be \(T\) then, to solve the
first network flow problem, we simply have to run a max-flow algorithm on a graph with
\(O(G + T)\) nodes and \(O(GT)\) edges.

Similarly to solve the second network flow problem, if we let the number of words be \(W\) we
simply have to run a max-flow algorithm on a graph with \(O(G+W)\) nodes and \(O(GW)\) edges.

Ultimately, the total running time depends on the algorithm employed to solve the max-flow
problems.

**Question 4 : Mi-k Drop**

Let \(G = (V, E)\) be a directed graph with a source \(s \in V\), a sink \(t \in V\), and where every edge \(e \in E\) has
capacity exactly 1. Let \(k\) be the value of the maximum flow in \(G\). Question: Given an arbitrary integer
\(1 \leq q \leq k\), can you always remove \(q\) edges from \(G\) so that in the resulting graph \(G'\) the value of the maximum
flow is \(k-q\)?

**Answer:**
The answer to the question is **Yes** and we proceed to prove it.

First consider all the vertices that are reachable from the source \(s\) in the residual graph \(G_f\).
Let $S = \{ v \in V : v \text{ is reachable from } s \text{ in } G_f \}$

Then clearly $(S, S - V)$ forms an $s - t$ cut in $G$. Define the edges that cross the cut:

Let $Z = \{ e = (u, v) \in E : u \in S \text{ and } v \notin S \}$

In the figure below we see that $S = \{ s \}$ and $Z = \{ (s, a), (s, b), (s, c) \}$:

Since the maximum flow in $G$ is $k$, then by the “max-flow min-cut” theorem there exists a cut $(S, V - S)$ with capacity $\text{cap}(S, S - V) = v(f) = k$. We know that the capacity of each edge is 1, so there must be $k$ edges that “connect” $S$ and $S - V$. More formally, the set $Z$ contains exactly $k$ edges i.e. $|Z| = k$.

Finally, given an arbitrary integer $1 \leq q \leq k$ then we can remove $q$ edges from the set $Z$. By doing this, the cut $(S, S - V)$ will remain a minimum cut since we are decreasing the new capacity by removing edges. The new capacity of the minimum cut is therefore: $\text{cap}(S, S - V) = k - q$. As a result, by the “max-flow min-cut” theorem it is possible to create a maximum flow equal to $v(f) = \text{cap}(S, S - V) = k - q$ by removing $q$ edges from $Z$.

**Question 5 : Problem 25 Chapter 7 Kleinberg-Tardos**

**Answer:** We create graph $G$ in order to model this problem as a circulation problem:

- Let $g_i$ be the vertex that corresponds to friend $i$.
- Denote the imbalance of person $i$ as $q_i$.
- Let the demand of the vertex $g_i$ be the imbalance $q_i$ of person $i$. Note, that the imbalance of a person is positive if the money that is owed to this person is larger than the money that this person owes.
- For each ordered pair of friends $(i, j)$, draw an edge $e = (g_i, g_j)$ if $a_{ij} > 0$, with capacity $\infty$
For illustration purposes, consider an example with 8 friends (demands/imbalances are shown as the $q$ of each node):

$$
\begin{align*}
g_1 & \quad q = -2 \\
g_2 & \quad q = -1 \\
g_3 & \quad q = -8 \\
g_4 & \quad q = -5 \\
g_5 & \quad q = 6 \\
g_6 & \quad q = 1 \\
g_7 & \quad q = 5 \\
g_8 & \quad q = 4 
\end{align*}
$$

Figure 1: Graph $G$

We then reduce this problem to a network flow problem by attaching a super source $s$ and a super sink $t$ to the original graph creating a new Graph $G'$. More specifically, we add edges $(s, g_i)$ with capacity $-q_i$ for all the nodes with negative imbalance and then we also add edges $(g_i, t)$ with capacity $q_i$ for all the nodes with positive imbalance.
Claim 9. The value of the maximum flow in $G'$ is equal to the sum of the positive imbalances of each vertex of $G$.

Even if this fact is obvious, it also follows from complete analogy of the argument used in the book on page 382.

Claim 10. Given a maximum flow in $G'$, then it is always possible to write a set of checks that constitutes a set of consistent reconciliations.

Proof. Using Claim 9, given a maximum flow, every edge $(s, g_i)$ in $G'$ will be saturated. Therefore, person $i$ can write a check to person $j$ equal to the value of the flow connecting $(g_i, g_j)$. By doing that each person with positive imbalance would pay an amount equal to his imbalance. Moreover, since this is a maximal flow, all edges $(t, g_i)$ will be saturated meaning that each person with negative imbalance would be paid an amount equal to his imbalance.

A flow of in $G'$ that solves this problem without considering the restriction of edges, needs to satisfy two properties, namely:

(a) Capacity Conditions: $\forall e \in E$ we have $0 \leq f_e \leq c_e$

(b) Conservation Conditions: $\forall v \in V$ we have $f_{in} - f_{out} = 0$

Constructing such a flow, $f$, in $G'$ that satisfies both $(a), (b)$ is trivial since we can simply send $a_{ij}$ units of flow across any $s-t$ path of the form $(s, g_i, g_j, t)$ if person $i$ owes $a_{ij}$ money to person $j$. Nevertheless, $f$ would not generally satisfy the condition that only
$n - 1$ edges are used.

This can be seen in the following example:

![Diagram of network flow](image)

Figure 3: This flow uses all of the edges of $G'$

So we propose an algorithm that transforms $f$ into a new flow $f'$ such that both properties $(a), (b)$ hold and $f'$ uses at most $n - 1$ edges. Next we use this flow to write down a consistent reconciliation of at most $n - 1$ checks.

More precisely, in order to find $f'$ given $G, f$ we follow the next algorithm:

**Algorithm 3 Solution to finding $f'$**

1. **function** FindFPrime
2. Start with the obvious flow $f$ that solves the problem by using all the edges of $G'$ with each edge carrying flow equal to $a_{ij}$
3. for each undirected cycle in $G$ do
4. Select the minimum flow edge, say $e_{\text{min}}$, that belongs in the cycle
5. Remove $e_{\text{min}}$ from the cycle
6. For every other edge $e$ in the cycle, if $e_{\text{min}}$ has the same direction as $e$ then
7. update flow for $e$: $f'(e) = f(e) - f(e_{\text{min}})$
8. For every other edge $e$ in the cycle, if $e_{\text{min}}$ has opposite direction as $e$ then
9. update flow for $e$: $f'(e) = f(e) + f(e_{\text{min}})$
10. end for
11. If edge $(g_i, g_j)$ in $G'$ exists, then person $i$ writes a check to
12. person $j$ with value equal to the flow of the edge $(g_i, g_j)$
13. end function
Proof of Correctness:
First it is easy to observe that upon termination our algorithm will have eliminated all the cycles in the graph $G'$. Since $G'$ is connected and contains no cycles, it must be a tree or a forest. The fact that $G'$ is a tree or a forest implies that $f'$ uses at most $n - 1$ edges. As a result the proposed reconciliation uses at most $n - 1$ checks.

Now we show that by performing this transformation on the cycles of the graph, the resulting flow, $f'$ satisfies property $(a)$, that for every vertex $v$ it holds that: $f'_\text{in} - f'_\text{out} = 0$. To show this fact we enumerate all the possible cases.

First let the edge with the minimum flow in a cycle be, $e_{\text{min}}$, and let any other edge of the cycle be $e$, then:

- $e_{\text{min}}$ and $e$ can have the same direction while $e_{\text{min}}$ is facing in the vertex $v$, while $e$ is facing out of the vertex $v$. In that case by removing $e_{\text{min}}$ we have:

  $$f'_\text{in} = f_\text{in} - f(e_{\text{min}})$$

  By adding $f(e_{\text{min}})$ to the value of the flow of $e$ we have:

  $$f'_\text{out} = f_\text{out} + f(e_{\text{min}})$$

  Therefore:

  $$f'_\text{in} - f'_\text{out} = (f_\text{in} - f(e_{\text{min}})) - (f_\text{out} + f(e_{\text{min}})) = f_\text{in} - f_\text{out} = 0$$

  A similar argument is used if $e_{\text{min}}$ and $e$ have the same direction and $e_{\text{min}}$ is facing out of the vertex while $e$ is facing in the vertex, therefore it is omitted.

- $e_{\text{min}}$ and $e$ have opposite direction and both are facing in the vertex. Then by removing $e_{\text{min}}$:

  $$f'_\text{in} = f_\text{in} - f(e_{\text{min}})$$

  But adding $f(e_{\text{min}})$ to the value of the flow of $e$:

  $$f'_\text{in} = f'_\text{in} + f(e_{\text{min}}) = f_\text{in} - f(e_{\text{min}}) + f(e_{\text{min}}) = f_\text{in}$$

  Hence:

  $$f'_\text{in} - f'_\text{out} = f_\text{in} - f_\text{out} = 0$$

- Finally, $e_{\text{min}}$ and $e$ have opposite direction and both are facing out of the vertex. Then by removing $e_{\text{min}}$:

  $$f'_\text{out} = f_\text{in} - f(e_{\text{min}})$$

  But adding $f(e_{\text{min}})$ to the value of the flow of $e$:

  $$f'_\text{out} = f'_\text{out} + f(e_{\text{min}}) = f_\text{out} - f(e_{\text{min}}) + f(e_{\text{min}}) = f_\text{out}$$

  Hence:

  $$f'_\text{in} - f'_\text{out} = f_\text{in} - f_\text{out} = 0$$
Therefore, by applying this transformation to all the edges of the cycle we see that the resulting flow $f'$ satisfies property $(a)$ i.e. the “Conservation Condition”.

Property $(b)$ i.e the “Capacity Condition” is always satisfied since the capacities of $G$ are infinite.

Finally, because $f'$ satisfies conditions $(a), (b)$ and uses at most $n - 1$, it must be a valid flow for the problem.

**Running Time:**

Since there are $n$ vertices in $G$, there is a total of $O(n)$ cycles. Finding a cycle using BFS takes $O(V + E) = O(n + E)$ time. Therefore by the loop that spans lines 4 − 10, the algorithm takes total time $O(n(n + E))$.

This transformation is also depicted below for the example we have introduced in this exercise:

![Graph showing the Cycle. Black edge has minimum flow](image)

![Final Graph with flow after removing edge](image)