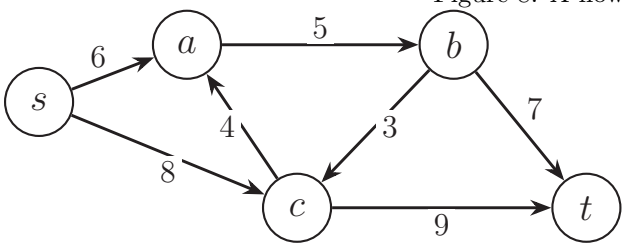
18 Max-Flow

In this chapter, we introduce a new problem to solve in directed graphs: Maximum flow. Like the shortest $s-t$ path problem, maximum flow asks about getting between two vertices s and t. In this problem, however, our goal is not to get between s and t as fast as possible, but to get as much "flow" as possible between s and t, across edges of limited capacity. This problem was originally studied to model the flow of goods from factories to consumers across rail networks, and remains relevant for many problems in logistics today.

18.1 Flows and Cuts

A **flow network** is a directed graph G with two special vertices s and t called the source and sink, respectively (s and t must be different vertices), where each edge e has an associated non-negative value $c(e)$ called the **capacity**.

Figure 8: A flow network



An **s-t-flow** (or simply a **flow**) is a function $f : E(G) \to \mathbb{R}_{\geq 0}$ which satisfies the following two constraints:

- $\forall\, e \in E(G), 0 \leq f(e) \leq c(e)$ (we call this the capacity constraint: the flow through an edge is at most its capacity)
- $\forall\, v \in V(G) \setminus \{s,t\}, \sum_{u \in N_{in}(v)} f(u,v) = \sum_{w \in N_{out}(v)} f(v,w)$ (we call this the flow conservation constraint: for every vertex besides the source and sink, the total flow into the vertex must equal the total flow out of the vertex).

For convenience we will often interpret f as a function from $V \times V$ to $\mathbb{R}_{\geq 0}$ with $f(u,v) = 0$ for every $(u,v) \notin E$.

The **value** of a flow f, denoted $|f|$, is the net flow out of s:

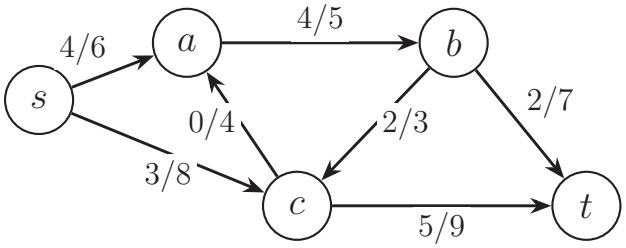
$$|f| := \left(\sum_{w \in N_{out}(s)} f(s,w) \right) - \left(\sum_{u \in N_{in}(s)} f(u,s) \right).$$

Note that by the conservation constraint, this is also equal to the net flow into t. The figure below illustrates a flow of value 7 in the network from Figure 8. Note that in this figure X/Y is not a fraction, it denotes sending X units of flow through an edge of capacity Y.

A **maximum flow** is a flow with maximum value across all flows in a network. The flow in Figure 9 is not a maximum flow (can you find a flow which achieves the actual maximum flow value of 13 for the network from Figure 8?). The central problem we will consider in this chapter is how to compute a max-flow. Before we can introduce an algorithm for this problem, however, it will be helpful to discuss cuts, which turn out to be closely related to flows.

Recall from Section 17.2 that a **cut** in an undirected graph is a partition of the vertices into two non-empty sets $S, V \setminus S$. Previously we discussed cuts for undirected graphs, where we said that an edge $\{u,v\}$ crosses the cut if one of u, v is in S and the other is in $V \setminus S$. We can define a related notion of cuts for flow networks.

Figure 9: A flow of value 7

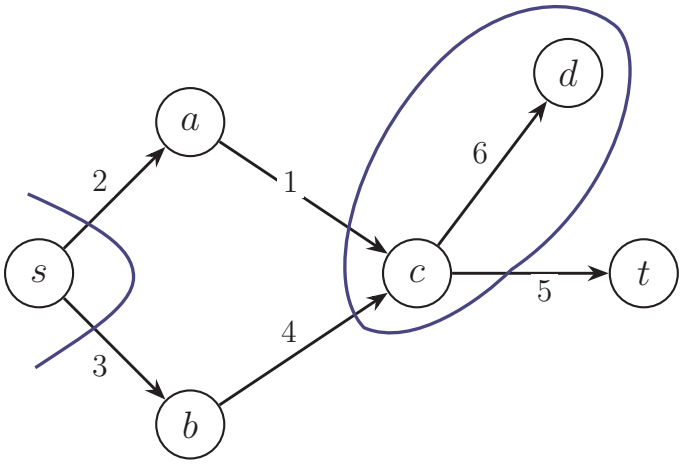


An s-t-**cut** (or simply a **cut**) in a flow network G is a partition of $V(G)$ into two sets S and T such that $s \in S$ and $t \in T$ (note that S, T being a partition means that we must have $S \cup T = V(G)$ and $S \cap T = \emptyset$). We say that an edge (u, v) **crosses** the cut if $u \in S$ and $v \in T$. Notably, we do *not* say that (u, v) crosses the cut if $u \in T$ and $v \in S$. We denote the set of edges crossing the cut S with $\delta_{out}(S)$.

The **value** (or **capacity**) of a cut S, T, denoted $c(S, T)$, is the total capacity of the edges crossing the cut. A cut with minimum value among all s-t-cuts in G is called a minimum s-t-cut (or simply min-cut).

The figure below depicts an s-t-cut with $S = \{s, c, d\}$ and $T = \{a, b, t\}$. Note that the definition of a cut does not require S or T to be connected. Since $s \in S, t \in T, S \cap T = \emptyset$, and $S \cup T = V$, this is a valid cut. Here $\delta_{out}(S) = \{(s, a), (s, b), (c, t)\}$. Note that (a, c) and (b, c) do not cross the cut. The value of this cut is $2 + 3 + 5 = 10$. This is not a min-cut in the network. The actual min-cut with a value of 4, is $S = \{s, a\}, T = \{b, c, d, t\}$.

Figure 10: An s-t-cut



18.2 Max-Flow Min-Cut

At first glance max-flow and min-cut may not seem related. They actually have a very close connection, however. For a flow f and a cut S, T we define the flow across the cut by

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

That is, $f(S, T)$ is the total flow from S to T minus the total flow from T back to S. The following theorem says that the flow through any s-t-cut is always equal to the value of the flow (recall that this is defined as the flow out of s).

Theorem 83. *Let G be a flow network, f be an s-t-flow in G, and S, T be an s-t-cut in G. Then $|f| = f(S,T)$.*

Proof.

$$\begin{aligned}
f(S,T) &= \sum_{u\in S, v\in T} f(u,v) - \sum_{u\in T, v\in S} f(u,v) \\
&= \sum_{u\in S, v\in T} f(u,v) - \sum_{u\in S, v\in T} f(v,u) \\
&= \sum_{u\in S, v\in T} f(u,v) + \sum_{u\in S, v\in S} f(u,v) - \sum_{u\in S, v\in T} f(v,u) - \sum_{u\in S, v\in S} f(v,u) \\
&= \sum_{u\in S}\left(\sum_{v\in T} f(u,v) + \sum_{v\in S} f(u,v) - \sum_{v\in T} f(v,u) - \sum_{v\in S} f(v,u)\right) \\
&= \sum_{u\in S}\left(\sum_{v\in V} f(u,v) - \sum_{v\in V} f(v,u)\right)
\end{aligned}$$

Note that $\sum_{v\in V} f(u,v)$ is the total flow out of vertex u and $\sum_{v\in V} f(v,u)$ is the total flow into vertex u. Thus, by our flow conservation constraint, for all $u \in V \setminus \{s,t\}$, we have $\sum_{v\in V} f(u,v) - \sum_{v\in V} f(v,u) = 0$. We know that $s \in S$ and $t \notin S$ by the definition of an s-t-cut, so therefore

$$\sum_{u\in S}\left(\sum_{v\in V} f(u,v) - \sum_{v\in V} f(v,u)\right) = \sum_{v\in V} f(s,v) - \sum_{v\in V} f(v,s) = |f|.$$

□

Theorem 83 implies the following corollary:

Theorem 84. *In any flow network G, the value of a maximum flow is less than or equal to the value of a minimum cut.*

Proof. Let f be a flow in G. By Theorem 83 for any cut $c(S,T)$,

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

In particular if f is a max-flow and S, T is a min-cut then $|f| \leq c(S,T)$. □

Theorem 84 tells us that we can upper bound the max-flow value in a network with the min-cut value. It turns out that these two values are actually *equal* as the following theorem states. This theorem is known as the max-flow min-cut theorem and was first proved by Lester Ford (same Ford who developed the Bellman-Ford algorithm) and Delbert Fulkerson in 1954.

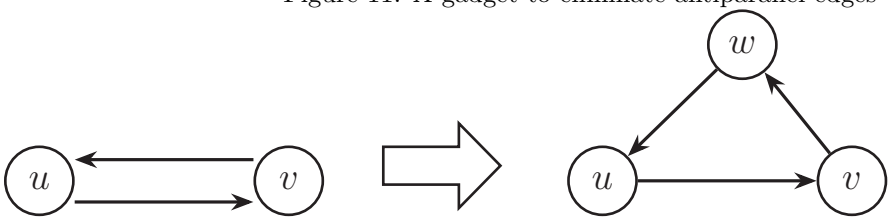
Theorem 85. *In every flow network with integer capacities, the value of a maximum s-t-flow is equal to the value of a minimum s-t-cut.*

Theorem 84 gives us half of this theorem. To finish the proof, we also need to show that the max flow value is *at least* the min-cut value. Ford and Fulkerson's approach to this was to give an algorithm which computes a flow with a value equal to the minimum cut value. We will present and analyze this algorithm in the next section. Theorem 85 in fact holds for arbitrary real capacities, but the proof is more involved and not important for this course.

18.3 Finding a Max-Flow: Ford-Fulkerson

In order to state Ford and Fulkerson's algorithm, we need a simplifying assumption and a few more definitions. The simplifying assumption we will use is that the input flow network G has no antiparallel edges, meaning that for every pair of vertices $u, v \in V$, at most one of the edges (u, v) and (v, u) is present in G. This is a safe assumption to make, because we can convert a flow network with antiparallel edges into an equivalent network which doesn't have them in $O(m + n)$. In particular, if we have a pair of vertices u, v such that (u, v) and (v, u) are both edges in G, we can introduce a new vertex w and replace the (v, u) edge with the length 2 path v, w, u. We set the capacity of both edges on this path equal to the capacity of (v, u) in the original graph. This transformation does not affect the maximum flow value since any flow which used the (v, u) edge can now use the v, w, u path and vice-versa. The transformation is illustrated in Figure 11.

Figure 11: A gadget to eliminate antiparallel edges



Henceforth we will assume that our input flow network has no antiparallel edges.

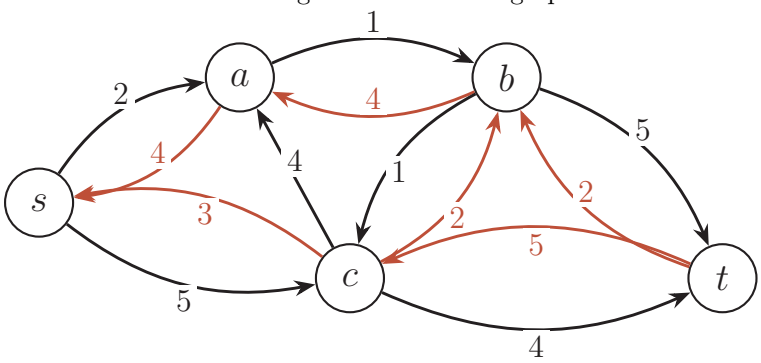
For a flow network G, and a flow f in G, we define the **residual capacity function** c_f by

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

Note that our assumption of no antiparallel edges guarantees that the first two cases are mutually exclusive (this is why we needed the assumption). We define the **residual graph** G_f for the flow f in the network G as the flow network on the same vertex set as G with the same source and sink, where the edge set is $E_f := \{(u, v) \mid u, v \in V(G) \wedge c_f(u, v) > 0\}$ and the capacity function is c_f.

Observe that the residual graph can have antiparallel edges even though the original network does not. Figure 12 illustrates the residual graph G_f for the flow from Figure 9. Edges that were part of the original graph G are depicted in black (note, however that their capacities have been reduced based on the amount of flow through them), while the new reverse edges are depicted in red.

Figure 12: Residual graph for the flow from Figure 9



We define an **augmenting path** for a flow f as an $s - t$ path in the residual graph G_f. The **residual capacity** of an augmenting path P is the amount of flow that we can push through that

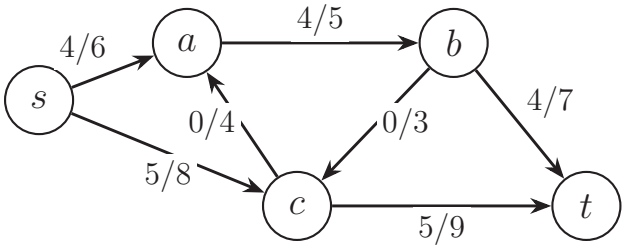
path in the residual graph: $c_f(P) := \min_{e \in P} c_f(e)$ (here we abuse notation slightly by viewing P as a set of edges rather than a sequence of vertices). For example, for the flow f from figure 9, s, c, b, t is an augmenting path. It has residual capacity 2, since that is the capacity in the residual graph of (c, b), the edge on this path with lowest residual capacity.

To **augment** f along a path P in the residual graph means doing the following for each edge (u, v) in P:

- If $(u, v) \in E(G)$, increase $f(u, v)$ by $c_f(P)$
- If $(v, u) \in E(G)$, decrease $f(v, u)$ by $c_f(P)$

For example, if we augment the flow from Figure 9 along the path s, c, b, t, we obtain the flow shown below:

Figure 13: A flow of value 9



It may not be clear that f remains a valid flow after augmenting. We will prove this now.

Theorem 86. *Let f be a flow and let f' be the result of augmenting f along an augmenting path P in the residual graph. Then f' is a valid flow of value $|f| + c_f(P)$.*

Proof. To show that f' is a valid flow we need to show that f' still satisfies the capacity and flow conservation constraints.

- First we show that f' satisfies the capacity constraints. Let $e = (u, v)$ be an arbitrary edge of G. We need to show that $0 \leq f'(e) \leq c(e)$. We consider three cases:
  - Suppose that $(u, v) \in P$. Then

    $$f'(u, v) = f(u, v) + c_f(P) > f(u, v) \geq 0$$

    and

    $$f'(u, v) = f(u, v) + c_f(P) \leq f(u, v) + c_f(u, v) = f(u, v) + c(u, v) - f(u, v) = c(u, v).$$

  - Suppose that $(v, u) \in P$. Then

    $$f'(u, v) = f(u, v) - c_f(P) \geq f(u, v) - c_f(v, u) = f(u, v) - f(u, v) = 0$$

    and

    $$f'(u, v) = f(u, v) - c_f(P) \leq f(u, v) \leq c(u, v).$$

  - Suppose that neither (u, v) nor (v, u) is in P. Then augmenting along P does not change the flow through (u, v), so $f'(u, v) = f(u, v)$ and so, since f is a valid flow, $0 \leq f'(e) \leq c(e)$.
- We need to show that for each vertex $v \in V(G) \backslash \{s, t\}$, $\sum_{u \in N_{in}(v)} f'(u, v) = \sum_{w \in N_{out}(v)} f'(v, w)$.If P does not visit v, then all of the edges incident to v have the same values under f' as under f so flow conservation is maintained. Otherwise, P must enter and exit v. We note that:

- If P enters v on an edge $(u, v) \in E(G)$ then f' increases the incoming flow to v along this edge by $c_f(P)$, while if P enters on an edge (w, v) with $(v, w) \in E(G)$, then f' decreases the outgoing flow along (v, w) by $c_f(P)$. Either way, the contribution from P's incoming edge causes v's in-flow to increase by $c_f(P)$ relative to its out-flow.
- If P exits v on an edge $(v, w) \in E(G)$, then f' increases the outgoing flow along (v, w) by $c_f(P)$, while if P exits on an edge (v, u) with $(u, v) \in E(G)$, then f' decreases the incoming flow along (u, v) by $c_f(P)$. Either way, the contribution from P's outgoing edge causes v's out-flow to increase by $c_f(P)$ relative to its in-flow.

These two effects cancel each other out, so the out-flow from v remains equal to the in-flow.

Finally, we note that f' increases the net flow out of s (and thus the value of the flow) by $c_f(P)$ by increasing the amount of flow through an edge (s, v) or by decreasing the amount of flow through an edge (v, s). □

Finally, we are ready to state the Ford-Fulkerson algorithm.

```
FORD-FULKERSON(G)
  f ← an empty flow (i.e. f(e) = 0   ∀ e ∈ E(G))
  While there exists an augmenting path P in the residual graph G_f:
      Augment f along P
  Return f
```

Theorem 87. *If G is a flow network with integer capacities,* FORD-FULKERSON(G) *returns a maximum flow in G.*

Proof. As shown in Theorem 86, every time we augment along an augmenting path, we increase the value of our flow by the residual capacity of the augmenting path. Since all of the capacities in G are integers, all of the capacities in G_f will be integers for any integer flow f. Since, by definition, G_f contains no 0 capacity edges, this means that if P has an augmenting path, that path has residual capacity at least 1. Thus, each iteration of the while loop, the value of our flow increase by at least 1. Since the maximum flow value in any network with integer capacities is some finite value, this means we must eventually reach a point where there are no augmenting paths and return f. We need to show that at this point f is a maximum flow. We will do this by showing that f has the same value as some cut. By Theorem 84 this implies that f must be a maximum flow (since no flow can have value greater than any cut).

Let f be the flow that FORD-FULKERSON returns. Since the residual graph contains no augmenting paths, we know that there is no s-t-path in G_f. Let $S := reach_{G_f}(s)$ and $T := V \setminus S$. None of the edges from S to T are present in G_f (otherwise we could use one of those edges to reach more vertices from s). By the definition of G_f, this means that for every edge $(u, v) \in E(G)$ with $u \in S, v \in T$ we have $f(u, v) = c(u, v)$. and for every edge $(u, v) \in E(G)$ with $u \in T, v \in S$, we have $f(u, v) = 0$. Therefore, $f(S, T) = c(S, T)$. By Theorem 83, $|f| = f(S, T)$, so by Theorem 84, f is a maximum flow. □

Note that Theorem 87 implies that every flow network with integer capacities has an integral maximum flow (i.e. we do not need to consider sending fractional amounts of flow through edges). The assumption that the edge capacities are integers is necessary in order to guarantee that the while loop terminates in finite time. With irrational capacities it is actually possible to create a graph where FORD-FULKERSON runs forever.

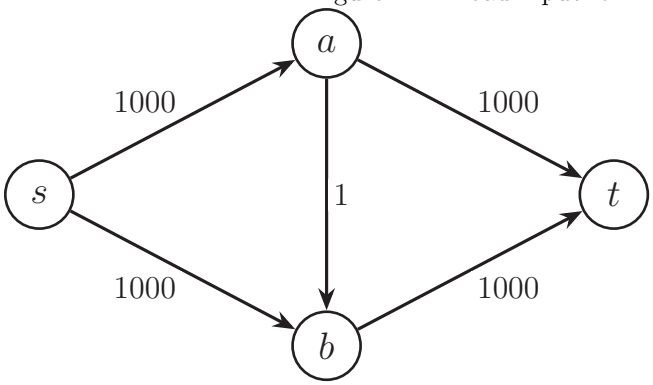
We note that Theorem 87 completes the proof of Theorem 85. The proof of Theorem 87 also gives us a way to efficiently find a min-cut, given a max flow. If f is a max-flow (computed by FORD-FULKERSON or any other algorithm), then for $S := reach_{G_f}(s)$, we have that $S, V \setminus S$ is a min s-t-cut.

18.4 Speeding Up Max-Flow: Edmonds-Karp

While Ford-Fulkerson will always find a maximum flow in a flow network with integer capacities, it may do so very slowly if we choose the wrong paths to augment along. For example, for the graph shown in Figure 14, if we alternate augmenting along s, a, b, t and augmenting along s, b, a, t, it will take 2000 rounds of augmentation to obtain the maximum flow in the graph, even though there are only 4 vertices in the network!

We can speed up the algorithm exponentially by being strategic about which paths we choose to augment along. This idea was developed by Jack Edmonds and Richard Karp in 1972.

Figure 14: A bad input for Ford-Fulkerson



Edmonds-Karp(G)
  $f \leftarrow$ an empty flow (i.e. $f(e) = 0 \quad \forall\, e \in E(G)$)
  While there exists an augmenting path in the residual graph G_f:
    Use BFS to find an augmenting path P with the fewest possible edges
    Augment f along P
  Return f

Since Edmonds-Karp is just a particular implementation of Ford-Fulkerson (rather than augmenting along an arbitrary augmenting path, we are augmenting along the shortest one) the correctness of this algorithm follows immediately from Theorem 87. The runtime analysis is a bit more complicated, but it is possible to prove the following theorem (see CLRS or Erickson for details):

Theorem 88. *If G is a flow network with integer capacities,* Edmonds-Karp(G) *outputs a maximum s-t-flow in G in $O(m^2 n)$ time.*

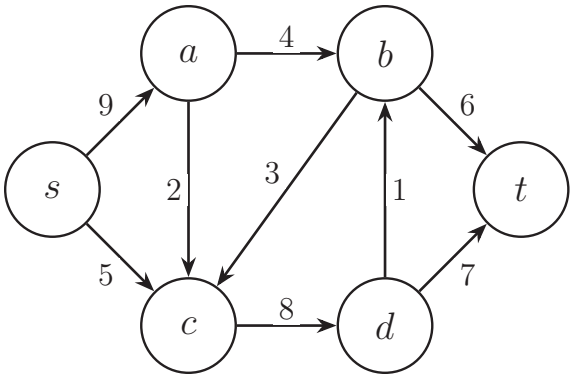
Over the 50 years since Edmonds and Karp described this algorithm in 1972, faster algorithms for max-flow were steadily developed, using increasingly complicated techniques, culminating in a 2022 algorithm by Chen, Kyng, Liu, Peng, Gutenberg, and Sachdeva which computes a maximum flow in a flow network with integer capacities in $O(m^{1+o(1)})$ time (almost linear!). As usual, you may assume a subroutine to solve max-flow in $O(m^{1+o(1)})$ time in your algorithms without needing to read and understand CKLPGS's 100 page paper.

18.5 Comprehension Questions

1. Suppose that a flow network has an $s \to t$ flow of value 100. What can we say about the min $s \to t$ cut in this network?

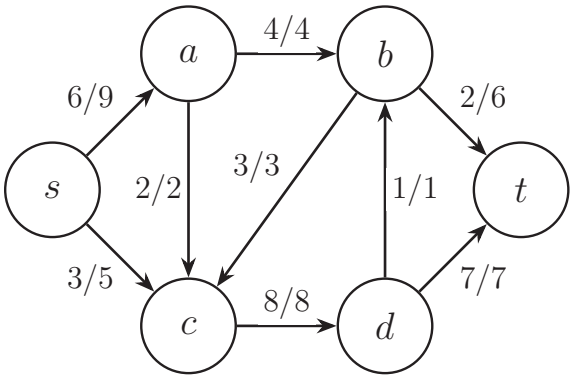
2. Describe a minimum s-t cut in the flow network below by giving the vertices which are part of S and those which are part of T.

   $S :=$ _____



$T :=$ _____

3. Below is a flow in the network from the previous problem. Give an example of an augmenting path in the residual graph for this flow.



Augmenting path: _____

4. What is the residual capacity of your augmenting path from the previous question?

Residual Capacity: _____

18.6 Exercises

1. Let G be a flow network, and suppose that (S_1, T_1) and (S_2, T_2) are two distinct min s-t-cuts in the network. Prove that $(S_1 \cup S_2, T_1 \cap T_2)$ is also a min s-t-cut in the flow network.

2. Let f and f' be s-t-flows with $|f| < |f'|$. Prove that there exists an s-t flow with value $|f'| - |f|$ in the residual graph G_f.

3. Let G be a flow network with integer edge capacities, let e be an edge of positive capacity in G, and let f be an integral max-flow for G (meaning f sends an integer amount of flow through every edge).

   - Give an $O(m+n)$-time algorithm which takes G, f, and e and computes the new maximum flow f' for the graph G' where e's capacity is increased by 1.

- Give an $O(m+n)$-time algorithm which takes G, f, and e and computes the new maximum flow f' for the graph G' where e's capacity is decreased by 1.

4. You are the owner of a large restaurant, and a group of customers has just come in asking to be served. Rather than ordering like normal customers, each of these customers has given you a list of dishes from your menu that they would be happy to eat. You want to serve each customer one of the dishes that customer has selected. In order to maintain the pristine reputation of your restaurant, you refuse to serve any two of the customers the exact same dish. Furthermore, each dish must be prepared by a chef who has truly mastered that dish, and each chef can prepare at most 4 dishes (to avoid making the customers wait too long). Suppose that you are given the following:

   - A number of customers: x,
   - A number of dishes on the menu: y,
   - A number of chefs: z,
   - A 2-D boolean array of customer preferences $Customer[1,\ldots,x][1,\ldots,y]$, where $Customer[i][j]$ is true if and only if the ith customer is happy to eat the jth dish on the menu,
   - A 2-D boolean array of chef specialties $Chef[1,\ldots,z][1,\ldots,y]$, where $Chef[i][j]$ is true if and only if the ith chef is able to prepare the jth dish on the menu.

   Give a polynomial time algorithm to determine whether or not it is possible to serve every customer a unique dish that they would be happy to eat while having each chef prepare at most 4 dishes and only prepare dishes they are able to make.