In these next three lectures we are going to talk about an important algorithmic problem called the *Network Flow Problem*. Network flow is important because it can be used to model a wide variety of different kinds of problems. So, by developing good algorithms for solving flows, we get algorithms for solving many other problems as well. In Operations Research there are entire courses devoted to network flow and its variants.

Objectives of this lecture

In this lecture, we will:

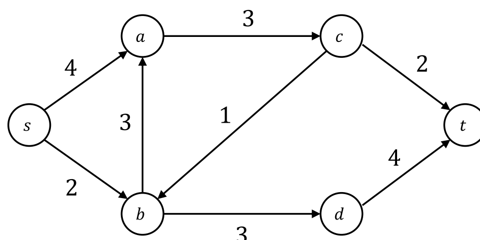
- Define the maximum network flow problem
- Derive and analyze the Ford-Fulkerson algorithm for maximum network flow
- Prove the famous *max-flow min-cut theorem*
- See how to solve the bipartite matching problem by reducing it to a maximum flow problem

Recommended study resources

- CLRS, *Introduction to Algorithms*, Chapter 26, Maximum Flow
- DPV, *Algorithms*, Chapter 7.2, Flows in Networks

1 The Maximum Network Flow Problem

We begin with a definition of the problem. We are given a directed graph G, a source node s, and a sink node t. Each edge e in G has an associated non-negative *capacity* $c(e)$, where for all non-edges it is implicitly assumed that the capacity is 0. For instance, imagine we want to route message traffic from the source to the sink, and the capacities tell us how much bandwidth we're allowed on each edge. For example, consider the graph below.



Our goal is to push as much *flow* as possible from s to t in the graph. The rules are that no edge can have flow exceeding its capacity, and for any vertex except for s and t, the flow *in* to the vertex must equal the flow *out* from the vertex. That is,

Definition: Flow constraints

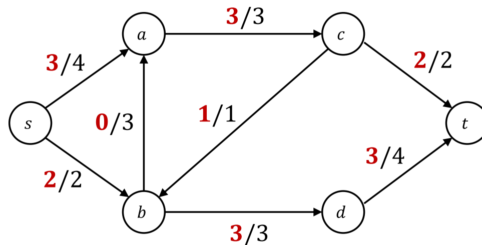
Capacity constraint: On any edge e we have $0 \leq f(e) \leq c(e)$.

Flow conservation: For any vertex $v \notin \{s,t\}$, flow in equals flow out: $\sum_u f(u,v) = \sum_u f(v,u)$.

A flow that satisfies these constraints is called a *feasible flow*. Subject to these constraints, we want to maximize the net flow from s to t. We can measure this formally by the quantity

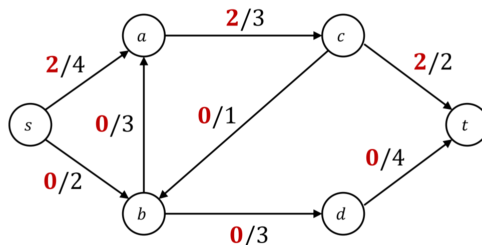
$$|f| = \sum_{u \in V} f(s,u) - \sum_{u \in V} f(u,s)$$

Note that what we are measuring here is the net flow coming out of s. It can be proved, due to conservation, that this is equal to the net flow coming into t. So, in the above graph, what is the maximum flow from s to t? Answer: 5. Using the notation"**flow**/capacity", a maximum flow looks like this. Note that the flow can split and rejoin itself.



1.1 Improving a flow: s-t paths

Suppose we start with the simplest possible flow, the all-zero flow. This flow is feasible since it meets the capacity constraints, and all vertices have flow in equal to flow out (zero!). But for the network above, the all-zero flow is clearly not optimal. Can we formally reason about *why* it isn't optimal? One way is to observe that there exists a path from s to t in the graph consisting of edges with available capacity (actually there are many such paths). For example, the path $s \to a \to c \to t$ has at least 2 capacity available, so we could add 2 flow to each of those edges without violating their capacity constraints, and improve the value of the flow. This gives us the flow below



An important observation we have just made is that if we add a constant amount of flow to all of the edges on a path, we never violate the flow conservation condition since we add the same amount of flow in and flow out to each vertex, except for s and t. As long as we do not violate the capacity of any edge, the resulting flow is therefore still a feasible flow.

This observation leads us to the following idea: A flow is not optimal if there exists an s-t path with available capacity. In the above graph, we can observe that the path $s \to b \to d \to t$ has 2 more units of available capacity, and hence add 2 units of flow to the edges. Lastly, we could find the $s \to a \to c \to b \to d \to t$ path has 1 more unit of available capacity, and we would arrive at the maximum flow from before.

1.2 Certifying optimality of a flow: s-t cuts

We just saw how to convince ourselves that a flow was **not** maximum, but how can you see that the above flow was really maximum? We can't be certain yet that we didn't make a bad decision and send flow down a sub-optimal path. Here's an idea. Notice, this flow saturates the $a \to c$ and $s \to b$ edges, and, if you remove these, you disconnect t from s. In other words, the graph has an "s-t cut" of capacity 5 (a set of edges of total capacity 5 such that if you remove them, this disconnects the sink from the source). The point is that any unit of flow going from s to t must take up at least 1 unit of capacity in these pipes. So, we know we're optimal. Let's formalize this idea of a "cut" and use it to formalize these observations.

Definition: s-t cut

An **s-t cut** is a set of edges whose removal disconnects t from s. Equivalently, a cut is a partition of the vertex set into two pieces S and T where $s \in S$ and $t \in T$. (The edges of the cut are then all edges going from S to T).

The **capacity** of a cut (S, T) is the sum of the capacities of the edges crossing the cut, i.e., the sum of the capacities of the edges going from S to T:

$$\text{cap}(S, T) = \sum_{u \in S} \sum_{v \in T} c(u, v)$$

Note importantly that the definition of the capacity of a cut **does not** include any edges that go from T to S, for example, the edge (c, b) above would not be included in the capacity of the cut $(\{s, a, b\}, \{c, d, t\})$.

Definition: Net flow

The **net flow** across a cut (S, T) is the sum of flows going from S to T minus any flows coming back from T to S, or

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

From this definition, we can see that the value of a flow $|f|$ is equivalent to the net flow across the cut $(\{s\}, V \setminus \{s\})$. In fact, using the flow conservation property, we can prove that the net flow across *any* cut is equal to the flow value $|f|$. This should make intuitive sense, because no matter where we cut the graph, there is still the same amount of flow making it from s to t.

Exercise

Prove the above statement, that the net flow across any (S, T) cut is equal to the value of the flow $|f|$. You should use the flow conservation constraint to do so. This also proves our earlier statement that the flow value $|f|$ is equal to the net flow into t.

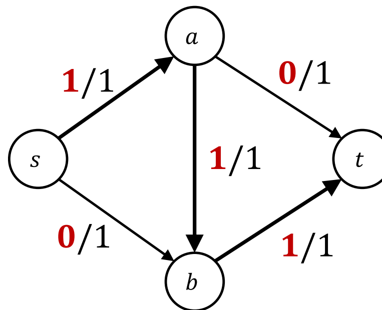
Now to formalize our observation from before, what we noticed is that the value of any s-t flow is less than the value of any s-t cut. Since every flow has a value that is at most the maximum flow, we can argue that in general, the maximum s-t flow $\leq$ the capacity of the minimum s-t cut, or

$$\text{the value of } \textit{any } s\text{-}t \text{ flow} \leq \text{maximum } s\text{-}t \text{ flow} \leq \text{minimum } s\text{-}t \text{ cut} \leq \text{capacity of } \textit{any } s\text{-}t \text{ cut}$$

This means that if we can find an s-t flow whose value is equal to the value of *any cut*, then we have proof that is must be a maximum flow!

1.3 Finding a maximum flow

How can we find a maximum flow and prove it is correct? We should try to tie together the two ideas from above. Here's a very natural strategy: find a path from s to t and push as much flow on it as possible. Then look at the leftover capacities (an important issue will be how exactly we define this, but we will get to it in a minute) and repeat. Continue until there is no longer any path with capacity left to push any additional flow on. Of course, we need to prove that this works: that we can't somehow end up at a suboptimal solution by making bad choices along the way. Is this the case for a naive algorithm that simply searches for s-t paths with available capacity and adds flow to them? Unfortunately it is not. Consider the following graph, where the algorithm has decided to add 1 unit of flow to the path $s \to a \to b \to t$



Are there any s-t paths with available capacity left? No, there are not. But is this flow a maximum flow? Also no, because we could have sent 1 unit of flow from $s \to a \to t$ and 1 unit of flow from $s \to b \to t$ for a value of 2. The problem with this attempted solution was that sending flow from a to b was a "mistake" that lowered the amount of available capacity left. To arrive at an optimal algorithm for maximum flow, we therefore need a way of "undoing" such mistakes. We achieve this by defining the notion of *residual capacity*, which accounts for both the remaining capacity on an edge, but also adds the ability to undo bad decisions and redirect a suboptimal flow to a more optimal one. This leads us to the Ford-Fulkerson algorithm.

2 The Ford-Fulkerson algorithm

The magic idea behind the Ford-Fulkerson algorithm is going to be "undoing" bad previous decisions by redirecting flow along a different path. To enable this, we define the concept of the *residual graph*. Residual capacity is just the capacity left over given the existing flow and also accounts for the ability to push existing flow back and along a different path, in order to undo previous bad decisions. Paths in the residual graph are called *augmenting paths*, because you use them to augment the existing flow.

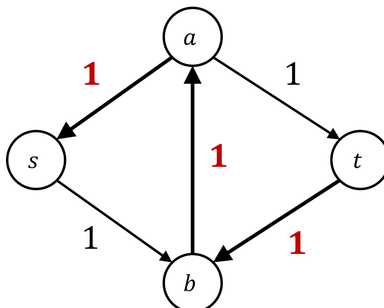
Definition: Residual capacity

Given a flow f in graph G, the **residual capacity** $c_f(u,v)$ is defined as

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

Definition: Residual graph

Given a flow f in graph G, the **residual graph** G_f is the directed graph with all edges of positive residual capacity, each one labeled by its residual capacity. Note: this may include reverse-edges of the original graph G.



When $(u,v) \in E$, the residual capacity on the edge corresponds to our existing intuitive notion of "remaining/leftover" capacity, it is the amount of additional flow that we could put through an edge before it is full. The second case is the key insight that makes the Ford-Fulkerson algorithm work. If $f(v,u)$ flow has been sent down some edge (v,u), then we are able to *undo* that by sending flow back in the reverse direction (u,v)! For example, if we consider the suboptimal flow from earlier, the residual graph looks like the following.

Unlike before, there is now an s-t path with capacity 1! The path $s \to b \to a \to t$ "undoes" the flow that was originally pushed through $a \to b$ and redirects it to $a \to t$, thus improving the flow and making it optimal.

The Ford-Fulkerson algorithm is then just the following procedure.

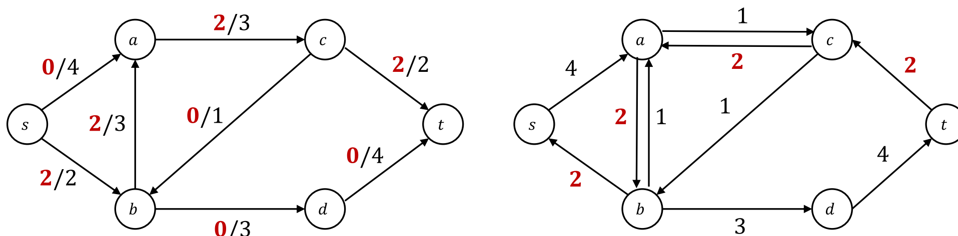
Algorithm: Ford-Fulkerson Algorithm for Maximum Flow

while (there exists an $s \to t$ path P of positive residual capacity)
  push the maximum possible flow along P

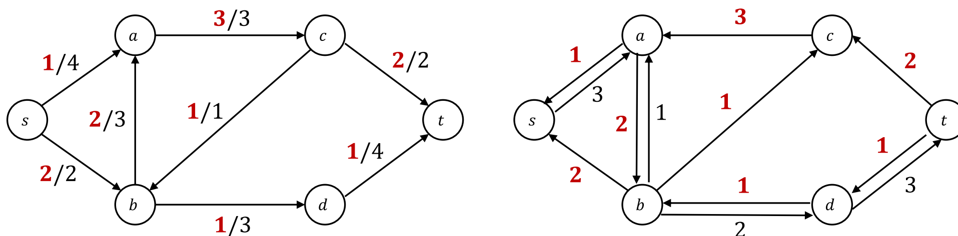
Remark: Handing anti-parallel edges

The definition of residual capacity becomes a little funny if we suppose that the graph contains anti-parallel edges. Recall that parallel edges are those with the same endpoints u and v that point in the same direction, while anti-parallel edges are those with the same endpoints but point in opposite directions. In this case, it might be the case that both $(u, v) \in E$ and $(v, u) \in E$, so how do we handle this? One option is to simply disallow it and not admit these kinds of graphs (this is what the textbook does!) Another valid answer is that we should use *both*. We have the choice of either combining both into a single edge with the sum of their capacities, or including a pair of parallel edges in the residual graph. Both of these options are fine and will work in theory and practice.

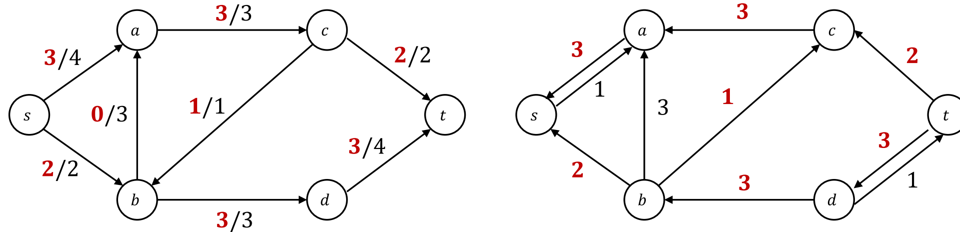
Let's see another example. Consider the graph we started with and suppose we push two units of flow on the path $s \to b \to a \to c \to t$. We then end up with the following flow (left) and residual graph (right).



If we continue running Ford-Fulkerson, we might find the *augmenting path* $s \to a \to c \to b \to d \to t$ which has capacity 1, bringing the flow value to 3, and yielding the following flow (left) and residual graph (right).



Finally, there is one more augmenting path, $s \to a \to b \to d \to t$ of capacity 2. This gives us the maximum flow that we saw earlier (left). At this point there is no longer a path from s to t in the residual graph (right) so we know we are done.



We can think of Ford-Fulkerson as at each step finding a new flow (along the augmenting path) and adding it to the existing flow. The definition of residual capacity ensures that the flow found by Ford-Fulkerson is *legal* (doesn't exceed the capacity constraints in the original graph). We now need to prove that in fact it is *maximum*. We'll worry about the number of iterations it takes and how to improve that later.

Note that one nice property of the residual graph is that it means that at each step we are left with same type of problem we started with. So, to implement Ford-Fulkerson, we can use any black-box path-finding method (e.g., DFS or BFS).

2.1 The Analysis

For now, let us assume that all the capacities are integers. If the maximum flow value is F, then the algorithm makes at most F iterations, since each iteration pushes at least one more unit of flow from s to t. We can implement each iteration in time $O(m+n)$ using DFS. If we assume the graph is simple, which is reasonable, $m \geq n-1$, so this is $O(m)$, so we get the following result.

Theorem: Runtime of the Ford-Fulkerson Algorithm

If the given graph G has integer capacities, Ford-Fulkerson terminates in time $O(mF)$ where F is the value of the maximum s-t flow.

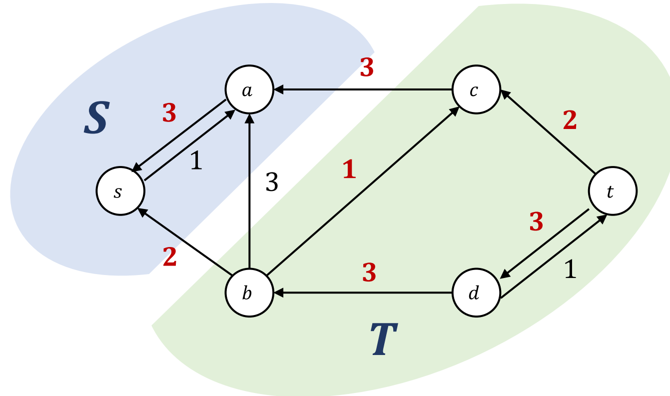
Now the important result that proves the correctness of the algorithm and comes with a powerful corollary.

Theorem: Optimality of the Ford-Fulkerson Algorithm

When it terminates, the Ford-Fulkerson algorithm outputs a flow whose value is equal to the minimum cut of the graph.

Proof. Let's look at the final residual graph. Since Ford-Fulkerson loops until there is no longer a path from s to t, this graph must have s and t disconnected, otherwise the algorithm would keep looping. Let S be the set of vertices that are still reachable from s in the residual graph, and let T be the rest of the vertices.

It must be that $t \in T$ since otherwise s would be connected to t, so this is a valid s-t cut. Let $c = \text{cap}(S,T)$, the capacity of the cut in the *original* graph. From our earlier discussion, we know that $|f| = f(S,T) \leq c$. The claim is that we in fact *did* find a flow of value c (which therefore implies it is maximum). Consider all of the edges of G (the original graph) that cross the cut in either direction. We consider both cases:



1. Consider edges in G that cross the cut in the $S \to T$ direction. We claim that all of these edges are at maximum capacity (the technical term is *saturated*). Suppose for the sake of contradiction that there was an edge crossing from S to T that was not saturated. Then, by definition of the residual capacity, the residual graph would contain an edge with positive residual capacity from S to T, but this contradicts the fact that T contains the vertices that we can not reach in the residual graph, since we would be able to use this edge to reach a vertex in T. Therefore, we can conclude that every edge crossing the cut from S to T is saturated.

2. Consider edges in G that cross the cut in the T to S direction. We claim that all such edges are *empty* (their flow is zero). Again, suppose for the sake of contradiction that this was not true and there is a non-zero flow on an edge going from T to S. Then by the definition of the residual capacity, there would be a reverse edge with positive residual capacity going from S to T. Once again, this is a contradiction because this would imply that there is a way to reach a vertex in T in the residual graph. Therefore, every edge crossing from T to S is empty.

Therefore, we have for every edge crossing the cut from S to T that $f(u, v) = c(u, v)$, and for every edge crossing from T to S that $f(u, v) = 0$, so the *net flow* across the cut is equal to the capacity of the cut c. Since every flow has a value that is at most the capacity of the minimum cut, this cut must in fact be the minimum cut, and the value of the flow is equal to it. □

As a corollary, this also proves the famous maximum-flow minimum-cut theorem.

Theorem: Max-flow min-cut theorem

In any flow network G, for any two vertices s and t, the maximum flow from s to t equals the capacity of the minimum (s, t)-cut.

Proof. For any integer-capacity network, Ford-Fulkerson finds a flow whose value is equal to the minimum cut. Since the value of any flow is at most the capacity of any cut, this must be a maximum flow. □

We can also deduce *integral-flow theorem*. This turns out to have some nice and non-obvious implications.

Theorem: Integral flow theorem

Given a flow network where all capacities are integer valued, there exists a maximum flow in which the flow on every edge is an integer.

Proof. Given a network with integer capacities, Ford-Fulkerson will only ever find integer-capacity augmenting paths and will never create a non-integer residual capacity. Therefore it finds an integer-valued flow, and this is maximum flow, so there always exists an integer-valued maximum flow. □

It is not necessarily the case that every maximum flow is integer valued, only that there exists one that is.

Exercise

Give an example of a flow network with all integer capacities and a maximum flow on that network which has a non-integer value on at least one edge.

Lastly, the proof also sneakily teaches us how to actually find a minimum cut, too! What great value.

Exercise: Finding a minimum cut

Explain how to construct a minimum s-t cut given a maximum s-t flow of a network. Hint: The answer is hidden in the proof above.

Remark: Min-cut max-flow for non-integer capacities

Technically, we only proved the min-cut max-flow theorem for integer capacities. What if the capacities are not integers? Firstly, if the capacities are rationals, then choose the smallest integer N such that $N \cdot c(u,v)$ is an integer for all edges (u,v). We see that at each step we send at least $1/N$ amount of flow, and hence the number of iterations is at most NF, where F is the value of the maximum s-t flow. (One can argue this by observing that scaling up all capacities by N will make all capacities integers, whence we can apply our above argument.) And hence we get min-cut max-flow for rational capacities as well.

What if the capacities are irrational? In this case Ford-Fulkerson may not terminate. And the solution it converges to (in the limit) may not even be the max-flow! (See here[a], here[b].) But the maxflow-mincut theorem still holds, even with irrational capacities. There are several ways to prove this; here's one. Suppose not, and suppose there is some flow network with the maxflow being $\epsilon > 0$ smaller than the mincut. Choose integer N such that $\frac{1}{N} \leq \frac{\epsilon}{2m}$, and round all capacities down to the nearest integer multiple of $1/N$. The mincut with these new edge capacities may have fallen by $m/N \leq \epsilon/2$, and the maxflow could be the same as the original flow network, but still there would be a gap of $\epsilon/2$ between maxflow and mincut in this rational-capacity network. But this is not possible, because used Ford-Fulkerson to prove maxflow-mincut for rational capacities in the previous paragraph.

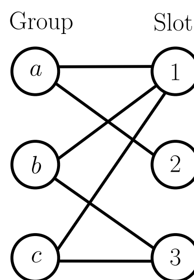
Alternatively, in the next lecture we'll see that if we modify Ford-Fulkerson to always choose an augmenting path with the fewest edges on it, it's guaranteed to terminate. This proves the max-flow min-cut theorem for arbitrary capacities.

[a]https://www.cs.princeton.edu/courses/archive/spring13/cos423/lectures/07DemoFordFulkersonPathological.pdf
[b]http://www.cs.huji.ac.il/ nati/PAPERS/ALGORITHMS/zwick.pdf

In the next lecture we will look at methods for reducing the number of iterations the algorithm can take. For now, let's see how we can use an algorithm for the max flow problem to solve other problems as well: that is, how we can *reduce* other problems to the one we now know how to solve.

3 Bipartite Matching

Say we wanted to be more sophisticated about assigning groups to homework presentation slots. We could ask each group to list the slots acceptable to them, and then write this as a bipartite graph by drawing an edge between a group and a slot if that slot is acceptable to that group. For example:



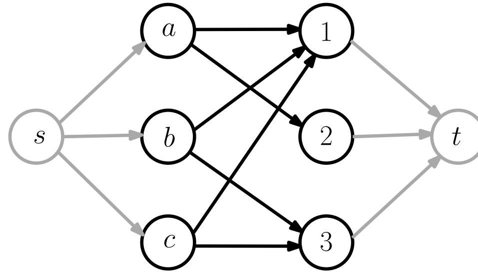
This is an example of a **bipartite graph**: a graph with two sides L and R such that all edges go between L and R. A **matching** is a set of edges with no endpoints in common. What we want here in assigning groups to time slots is a **perfect matching**: a matching that connects every point in L with a point in R. For example, what is a perfect matching in the bipartite graph above?

More generally (say there is no perfect matching) we want a **maximum matching**: a matching with the maximum possible number of edges. We can solve this as follows:

Algorithm: Bipartite Matching

1. Set up a source node s connected to all vertices in L. Connect all vertices in R to a sink node t. Orient all edges left-to-right and give each a capacity of 1.
2. Find a max flow from s to t using Ford-Fulkerson.
3. Output the edges between L and R containing nonzero flow as the desired matching.

Let's run the algorithm on the above example. We first build this flow network.



Then we use Ford-Fulkerson. Say we start by pushing flow on s-a-1-t and s-c-3-t, thereby matching a to 1 and c to 3. These are bad choices, since with these choices b cannot be matched. But the augmenting path s-b-1-a-2-t *automatically* undoes them as it improves the flow! Amazing.

Matchings come up in many different problems like matching up suppliers to customers, or cell-phones to cell-stations when you have overlapping cells. They are also a building block of other algorithmic problems.

Proof of correctness We will prove correctness in two parts. First we will show that for any valid matching, there exists a corresponding feasible flow, and for any feasible flow, there exists a valid matching.

First, consider any valid matching, and create a flow by assigning a flow value of 1 to each edge in the matching, and a flow value of 1 to each s and t edge adjacent to a matched edge. We claim that this flow is feasible. The capacity constraints are satisfied because we assign a flow of at most 1 per edge. What about conservation? We assign a flow of 1 only to s edges and t edges that are adjacent to a matched edge, and furthermore, since it is a valid matching, every matched vertex has one and only one matching edge, hence there is exactly 1 flow in and 1 flow out of every matched vertex, so the conservation constraint is satisfied.

Now consider any feasible flow, and construct a matching by taking all of the edges in the middle with flow value of 1. We claim this is a valid matching. Why? Because of the capacity constraint, each matched vertex has at most 1 flow coming in. Then because of the conservation constraint, it has at most 1 flow going out, so no vertex is matched multiple times.

Lastly, note that in both of the above correspondences, the cardinality of the matching is equal to the value of the flow, hence the value of the maximum flow is equal to cardinality of the maximum matching.

Runtime What about the number of iterations of Ford-Fulkerson? This is at most the number of edges in the matching since each augmenting path gives us one new edge. There can be at most n edges in the matching, so we have $F \leq n$, and hence the runtime is $O(nm)$.