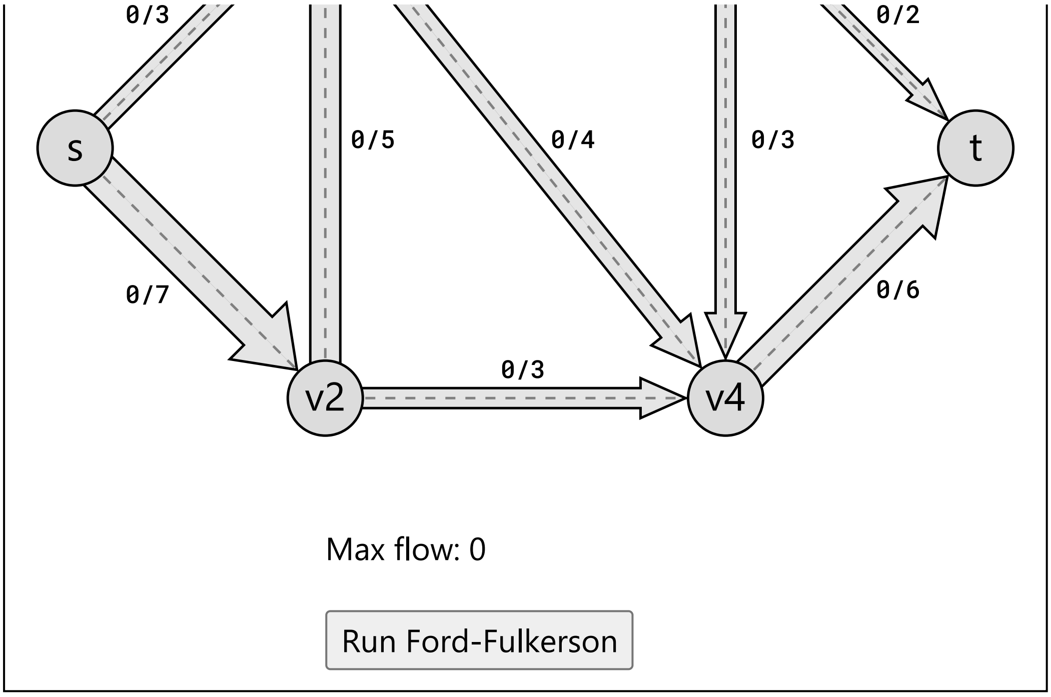
DSA Maximum Flow

❮ Previous Next ❯

The Maximum Flow Problem

The Maximum Flow problem is about finding the maximum flow through a directed graph, from one place in the graph to another.

More specifically, the flow comes from a source vertex s, and ends up in a sink vertex t, and each edge in the graph is defined with a flow and a capacity, where the capacity is the maximum flow that edge can have.



Finding the maximum flow can be very useful:

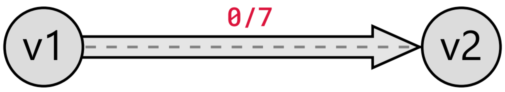
- For planning roads in a city to avoid future traffic jams.
- To assess the effect of removing a water pipe, or electrical wire, or network cable.
- To find out where in the flow network expanding the capacity will lead to the highest maximum flow, with the purpose of increasing for example traffic, data traffic, or water flow.

Terminology And Concepts

A **flow network** if often what we call a directed graph with a flow flowing through it.

The **capacity** c of an edge tells us how much flow is allowed to flow through that edge.

Each edge also has a **flow** value that tells how much the current flow is in that edge.



The edge in the image above $v_1 \rightarrow v_2$, going from vertex v_1 to vertex v_2, has its flow and capacity described as `0/7`, which means the flow is `0`, and the capacity is `7`. So the flow in this edge can be increased up to 7, but not more.

For all vertices except s and t, there is a **conservation of flow**, which means that the same amount of flow that goes into a vertex, must also come out of it.

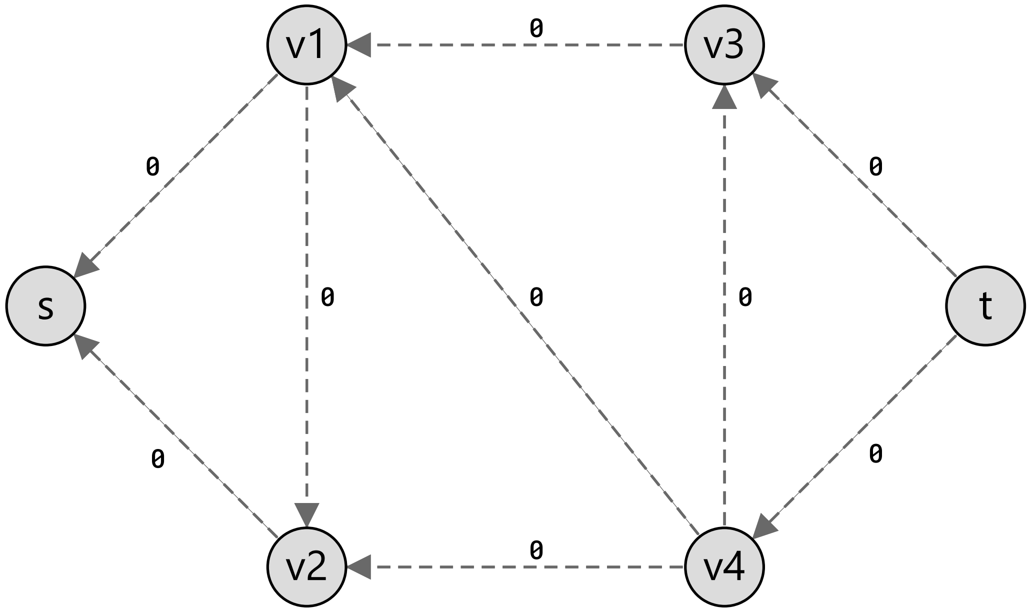
The maximum flow is found by algorithms such as Ford-Fulkerson, or Edmonds-Karp, by sending more and more flow through the edges in the flow network until the capacity of the edges are such that no more flow can be sent through. Such a path where more flow can be sent through is called an **augmented path**.

The Ford-Fulkerson and Edmonds-Karp algorithms are implemented using something called a **residual network**. This will be explained in more detail on the next pages.

The **residual network** is set up with the **residual capacities** on each edge, where the residual capacity of an edge is the capacity on that edge, minus the flow. So when flow is increased in a an edge, the residual capacity is decreased with the same amount.

For each edge in the residual network, there is also a **reversed edge** that points in the opposite direction of the original edge. The residual capacity of a reversed edge is the flow of the original edge. Reversed edges are important for sending flow back on an edge as part of the maximum flow algorithms.

The image below shows the reversed edges in the graph from the simulation at the top of this page. Each reversed edge points in the opposite direction, and because there is no flow in the graph to begin with, the residual capacities for the reversed edges are 0.



Some of these concepts, like the residual network and the reversed edge, can be hard to understand. That is why these concepts are explained more in detail, and with examples, on

Multiple Source and Sink Vertices

The Ford-Fulkerson and Edmonds-Karp algorithms expects one source vertex and one sink vertex to be able to find the maximum flow.

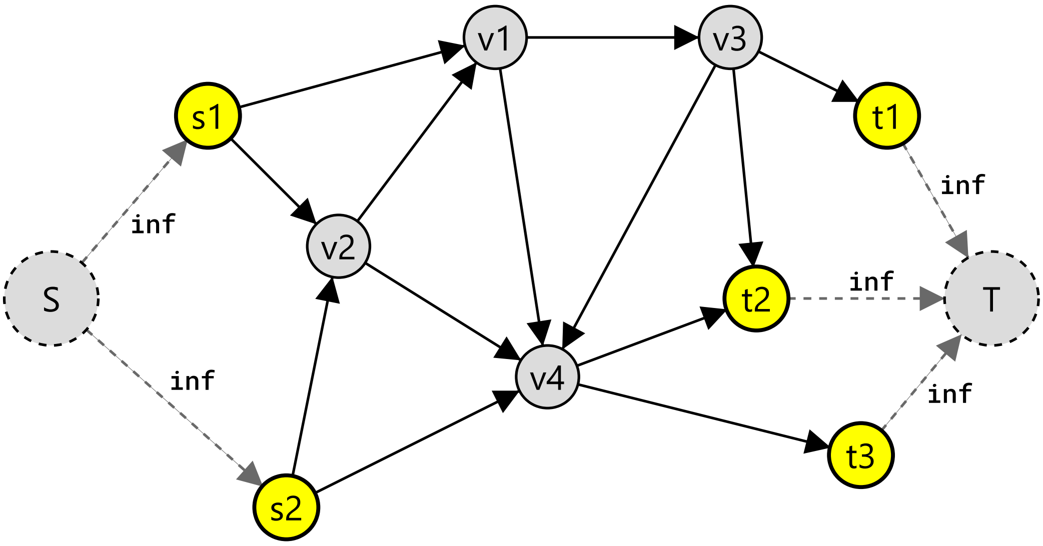
If the graph has more than one source vertex, or more than one sink vertex, the graph should be modified to find the maximum flow.

To modify the graph so that you can run the Ford-Fulkerson or Edmonds-Karp algorithm on it, create an extra super-source vertex if you have multiple source vertices, and create an extra super-sink vertex if you have multiple sink-vertices.

From the super-source vertex, create edges to the original source vertices, with infinite capacities. And create edges from the sink vertices to the super-sink vertex similarly, with infinite capacities.

The image below shows such a graph with two sources s_1 and s_2, and three sinks t_1, t_2, and t_3.

To run Ford-Fulkerson or Edmonds-Karp on this graph, a super source S is created with edges with infinite capacities to the original source nodes, and a super sink T is created with edges with infinite capacities to it from the original sinks.



The Ford-Fulkerson or Edmonds-Karp algorithm is now able to find maximum flow in a graph with multiple source and sink vertices, by going from the super source S, to the super

The Max-flow Min-cut Theorem

To understand what this theorem says we first need to know what a cut is.

We create two sets of vertices: one with only the source vertex inside it called "S", and one with all the other vertices inside it (including the sink vertex) called "T".

Now, starting in the source vertex, we can choose to expand set S by including adjacent vertices, and continue to include adjacent vertices as much as we want as long as we do not include the sink vertex.

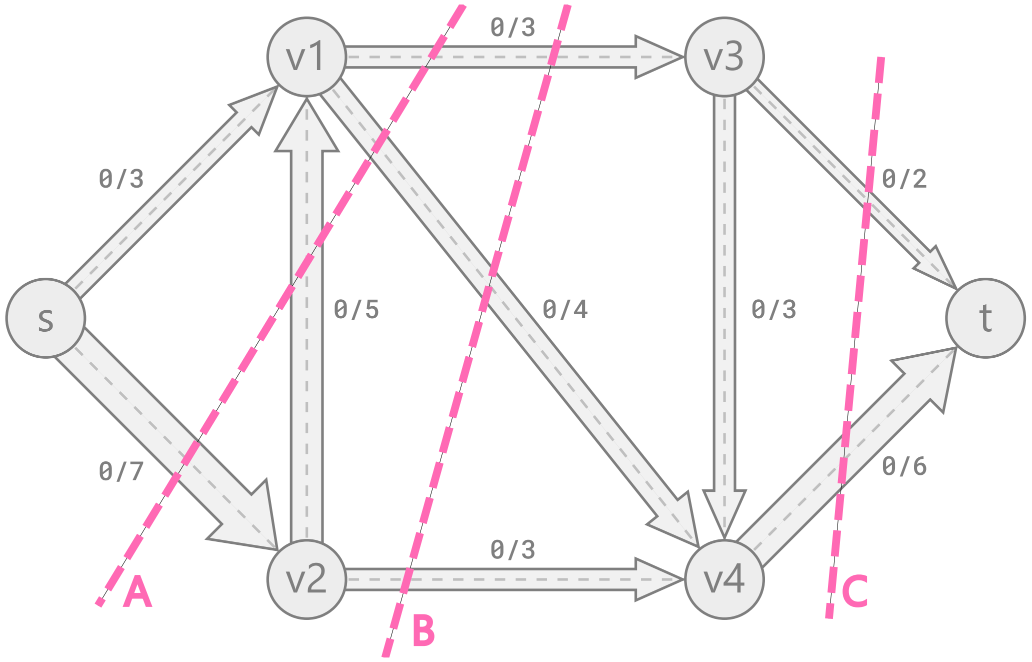
Expanding set S will shrink set T, because any vertex belongs either to set S or set T.

In such a setup, with any vertex belonging to either set S or set T, there is a "cut" between the sets. The cut consists of all the edges stretching from set S to set T.

If we add all the capacities from edges going from set S to set T, we get the capacity of the cut, which is the total possible flow from source to sink in this cut.

The minimum cut is the cut we can make with the lowest total capacity, which will be the bottleneck.

In the image below, three different cuts are done in the graph from the simulation in the top of this page.



Cut A: This cut has vertices s and v_1 in set S, and the other vertices are in set T. The total capacity of the edges leaving set S in this cut, from sink to source, is 3+4+7=14. We are not

Cut C: The maximum possible flow is 2+6=8 across cut C. If we checked all other cuts in the graph, we would not find a cut with a lower total capacity. This is the minimum cut. Have you run the simulation finding the maximum flow in the top of this page? Then you also know that 8 is the maximum flow, which is exactly what the max-flow min-cut theorem says.

The max-flow min-cut theorem says that finding the minimum cut in a graph, is the same as finding the maximum flow, because the value of the minimum cut will be the same value as the maximum flow.

Practical Implications of The Max-flow Min-cut Theorem

Finding the maximum flow in a graph using an algorithm like Ford-Fulkerson also helps us to understand where the minimum cut is: The minimum cut will be where the edges have reached full capacity.

The minimum cut will be where the bottleneck is, so if we want to increase flow beyond the maximum limit, which is often the case in practical situations, we now know which edges in the graph that needs to be modified to increase the overall flow.

Modifying edges in the minimum cut to allow more flow can be very useful in many situations:

- Better traffic flow can be achieved because city planners now know where to create extra lanes, where to re-route traffic, or where to optimize traffic signals.
- In manufacturing, a higher production output can be reached by targeting improvements where the bottleneck is, by upgrading equipment or reallocating resources for example.
- In logistics, knowing where the bottleneck is, the supply chain can be optimized by changing routes, or increase capacity at critical points, ensuring that goods are moved more effectively from warehouses to consumers.

So using maximum flow algorithms to find the minimum cut, helps us to understand where the system can be modified to allow an even higher throughput.

The maximum flow problem is not just a topic in Computer Science, it is also a type of Mathematical Optimization, that belongs to the field of Mathematics.

In case you want to understand this better mathematically, the maximum flow problem is described in mathematical terms below.

All edges (E) in the graph, going from a vertex (u) to a vertex (v), have a flow (f) that is less than, or equal to, the capacity (c) of that edge:

$$\forall (u,v) \in E : f(u,v) \leq c(u,v)$$

This basically just means that the flow in an edge is limited by the capacity in that edge.

Also, for all edges (E), a flow in one direction from u to v is the same as having a negative flow in the reverse direction, from v to u:

$$\forall (u,v) \in E : f(u,v) = -f(v,u)$$

And the expression below states that conservation of flow is kept for all vertices (u) except for the source vertex (s) and for the sink vertex (t):

$$\forall u \in V \setminus \{s,t\} \Rightarrow \sum_{w \in V} f(u,w) = 0$$

This just means that the amount of flow going into a vertex, is the same amount of flow that comes out of that vertex (except for the source and sink vertices).

And at last, all flow leaving the source vertex s, must end up in the sink vertex t:

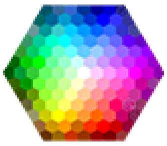
$$\sum_{(s,u) \in E} f(s,u) = \sum_{(v,t) \in E} f(v,t)$$

The equation above states that adding all flow going out on edges from the source vertex will give us the same sum as adding the flow in all edges going into the sink vertex.

❮ Previous Next ❯

Sign in to track progress

COLOR PICKER



PLUS SPACES

GET CERTIFIED FOR TEACHERS

Tutorials ▾ References ▾ Exercises ▾ Sign In

SS JAVASCRIPT SQL PYTHON JAVA PHP HOW TO W3.CSS C C

Top Tutorials

HTML Tutorial
CSS Tutorial
JavaScript Tutorial
How To Tutorial
SQL Tutorial
Python Tutorial
W3.CSS Tutorial
Bootstrap Tutorial
PHP Tutorial
Java Tutorial
C++ Tutorial
jQuery Tutorial

Top References

HTML Reference
CSS Reference
JavaScript Reference
SQL Reference
Python Reference
W3.CSS Reference
Bootstrap Reference
PHP Reference
HTML Colors
Java Reference
AngularJS Reference
jQuery Reference

Top Examples

HTML Examples
CSS Examples
JavaScript Examples
How To Examples
SQL Examples
Python Examples
W3.CSS Examples
Bootstrap Examples
PHP Examples
Java Examples
XML Examples
jQuery Examples

Get Certified

HTML Certificate
CSS Certificate
JavaScript Certificate
Front End Certificate
SQL Certificate
Python Certificate
PHP Certificate
jQuery Certificate
Java Certificate
C++ Certificate
C# Certificate
XML Certificate

FORUM ABOUT ACADEMY