**Incremental Improvement**

Table of Contents

[Ford-Fulkerson Algorithm 2](#_Toc63011455)

[Flow Networks 2](#_Toc63011456)

[Maximum Flow Problem 3](#_Toc63011457)

[Assumptions 4](#_Toc63011458)

[Defining Flows 5](#_Toc63011459)

[Flow Properties 6](#_Toc63011460)

[Cuts 8](#_Toc63011461)

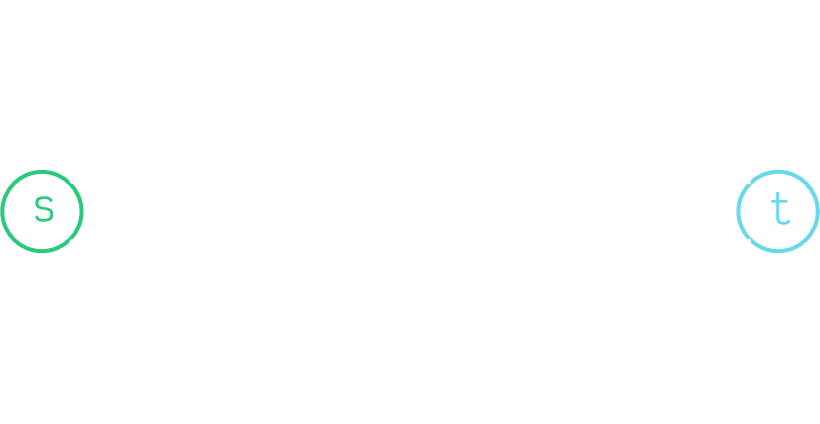
[Residual Networks 11](#_Toc63011462)

[Augmented Paths 13](#_Toc63011463)

## Ford-Fulkerson Algorithm

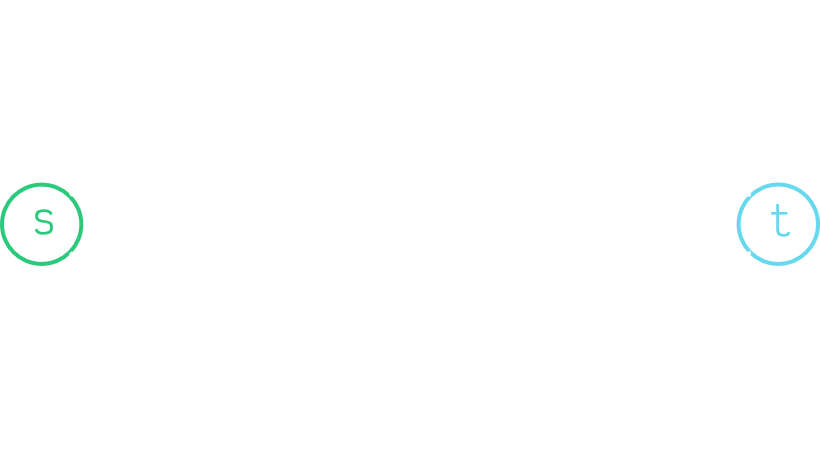
### Flow Networks

A flow network is basically a directed graph, , with two special vertices, the source, , and the sink, .



Essentially, there will be a ‘flow’ from the source to the sink. This flow will have to abide by certain constraints.

If we consider something like a water flow, then the flow network might have some capacity limitations. For example, consider the graph below.

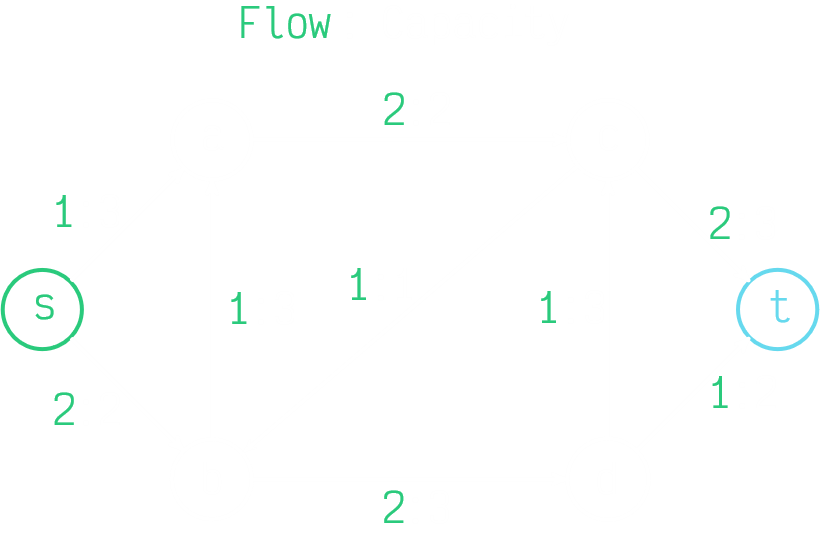


The edge weights denote the maximum flow capacity of the corresponding edges. Thus, for, say, the edge from to , only units of water can flow. Thus, every edge has some capacity, .

If there is no edge between two vertices, we can consider that the capacity for that (non-existent) edge is , i.e. if , . We will not be needing this information for now.

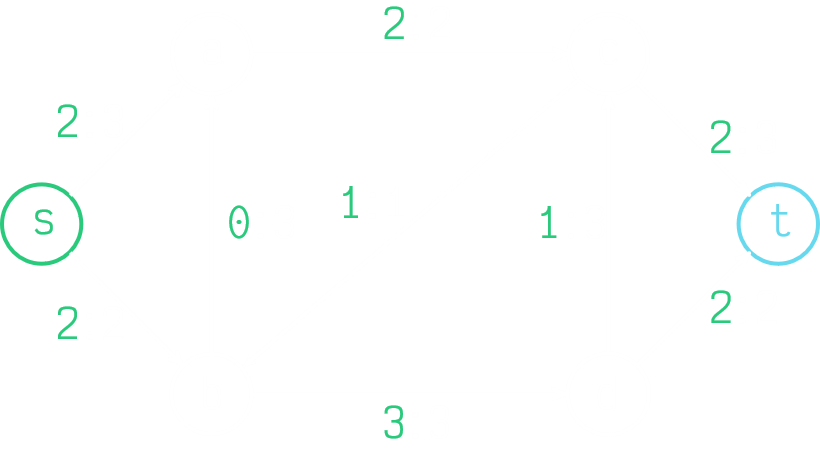
### Maximum Flow Problem

The problem is that we will start some flow from and get it to , while abiding by the capacity limitations of the edges we use. We can denote this as . The flow through an edge cannot exceed the capacity.



Notice that there is flow conservation, meaning the incoming flow to a vertex is the same as the outgoing flow. Other than the source and the sink, no other vertex accumulates or creates any flow. For example, flow goes into the vertex from two different edges and goes out using a third edge.

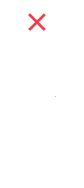
Our goal is to maximize the flow that can be given out by the source. The graph given just above this is not actually the one with the maximum flow. The maximum flow for this graph is .



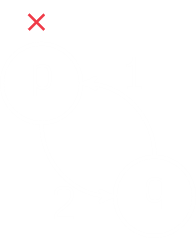
Consider how we managed this change. We decreased the flow from to from to and increased the flow from to from to . This allowed us to increase the flow from to from to , which resulted in an increased output from . The increased output was felt in the flow from to , which increased from to .

### Assumptions

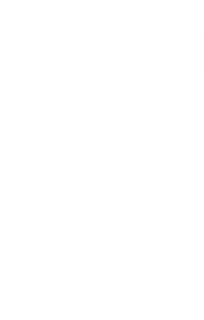
1. There will not be any self-loops.



1. If there is an edge from to , there cannot be an edge from to . This assumption is not made everywhere, but making it makes our life easier.



If we do find ourselves having to deal with a situation like this, we can add a vertex and change the graph so as to maintain that assumption.



### Defining Flows

A flow is defined as a function, , that has two vertices as its parameters and gives a real number as its result. A flow must satisfy three constraints:

1. **Capacity Constraint**

For all pairs of vertices , the flow from to , , is less than or equal to the capacity of the edge, , i.e. .

1. **Flow Conservation Constraint**

For all vertices other than the source and the sink, the total incoming flow, denoted by positive values, must be equal to the total outgoing flow, denoted by negative values. This results in the overall flow for the vertex being .

Formally, for all , .

1. **Skew Symmetry Constraint**

If we have a flow going from to , if we need to denoted the flow in the opposite direction, i.e. from to , it will be the negative of the same value.

The value of the flow is denoted by or . For the entire network, the flow depends on what the outgoing flow from the source is, since no other vertex can create any flow. Thus,

Using a capital indicates that the sum of the flows to all vertices is being considered. This is called the implicit summation notation. It will be used throughout this topic.

### Flow Properties

This can be proven using the skew symmetry constraint.

Essentially, taking a flow from to and sending it back from to gives us an overall flow of .

Notice that capital s where used, meaning instead of having a single vertex, we can also have a set of vertices and the result would be the same.

This is just the skew symmetric constraint, but for a set of vertices instead of a single one. For example,

1. If ,

Essentially, this means that if there are two sets of vertices that have no vertices in common, then the total flow from both sets to a third set will be the sum of the individual flows from the two sets.

Using these properties, we can now prove a theorem. We have already stated that the total flow in the network comes from the source. Now, we want to prove that

meaning the total flow of the network is equal to the sum of the flows from all possible vertices to the sink.

We know,

Additionally, if we consider all vertices that are not , i.e. , then, using the third property,

using the second property

using skew symmetry

From , if we take out ,

using skew symmetry

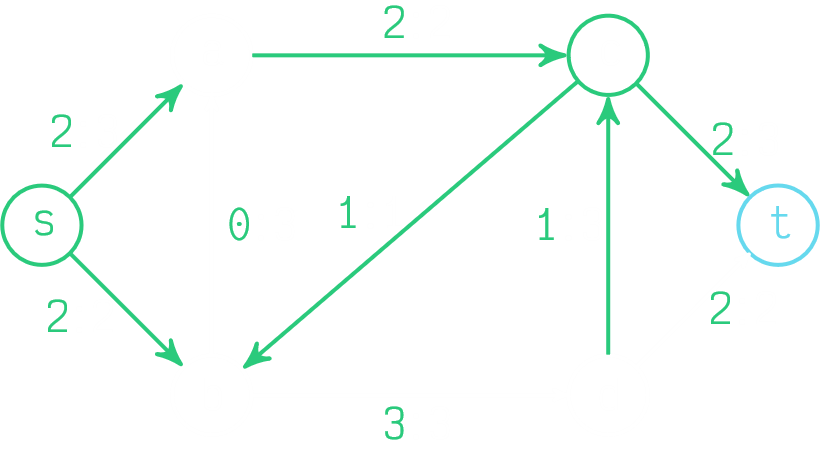
Since no vertex other than can create any flow and no vertex other than can accumulate any flow, taking out both and from means that no flow is created or accumulated. Thus,

using flow conservation

### Cuts

In flow networks, a cut is defined as a partition of two sets, . The partition is made such that the source vertex, , is always in , and the sink vertex, , is always in . All the other vertices can be in either set.

We are now concerned about the flow across the cut, which is the total flow going from to . Since is in and is in , we know that all of the flow in the network must become involved.



For example, in the graph above, say and are in while all the other vertices are in . Thus, the flow across the cut is

Notice that we are considering both incoming and outgoing flows.

#### Proof

We made a claim above, that since is in and is in , no matter how we partition the other vertices, as long as the two sets are disjoint sets, the flow across the cut will be the same as the flow for the entire network, i.e.

Since , we can say that . Thus,

using property 1

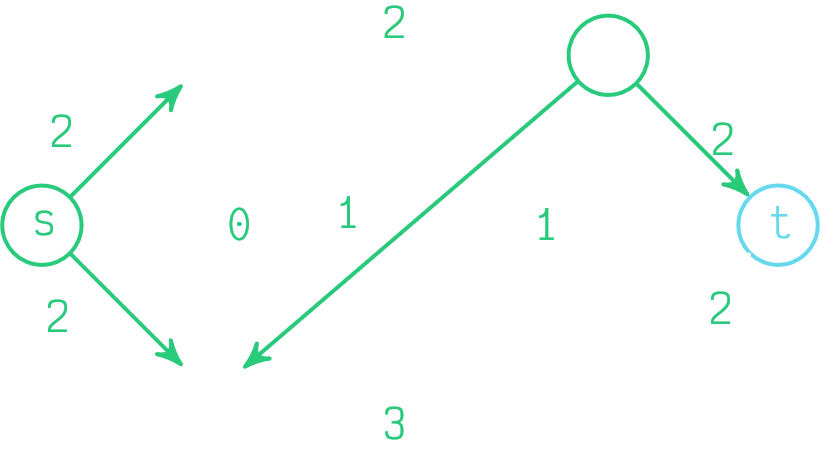
extracting from

We also know that cannot contain . If it does not contain either, due to flow conservation, there cannot be any flow originating at . Thus,

Finally, we already know that . Thus,

#### Cut Capacities

The capacity of a cut, , is defined as the summation of the capacities of the outgoing edges in .



For the graph above, the capacity is .

Understand that the flow across the cut is bounded by this capacity. The outgoing flow from cannot in any way exceed the capacity of the cut.

The reason this property is important is because the capacity of the cut will change depending on how we make the cut. For this cut, the capacity was , but for some other cut, we could get a capacity of or .

For some particular cut, we will get a capacity that is exactly the same as the maximum flow of our network, in this case. This cut is called the min cut.

### Residual Networks

A residual network is a copy of the original graph, meaning we still have all the edges and vertices. It is denoted as . The only difference is that we change the weights of the edges in an attempt to increase the flow. Thus, a residual network helps us figure out places in our graph where the flow can be increased.

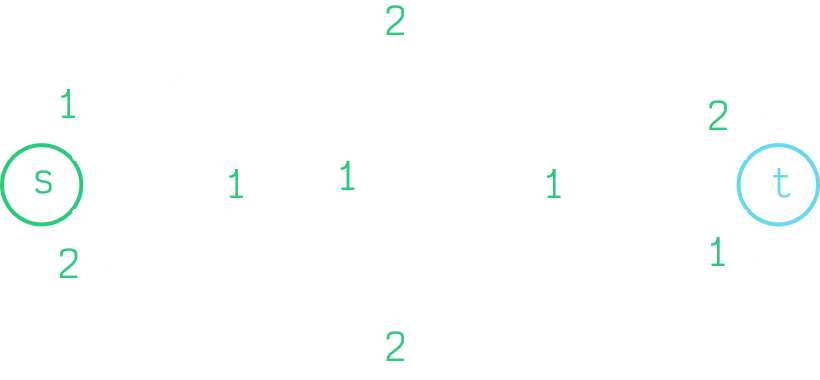
The rule to follow with residual networks is that there cannot be any edges with weights that are . Thus, if we have a particular edge in the original graph from to such that , then we will place an edge there as .

The edges allow a greater flow. For these edges, we can increase or decrease the flow values to see an overall increase in flow in the network.

Remember two properties in particular. Firstly, if , then . Secondly, .

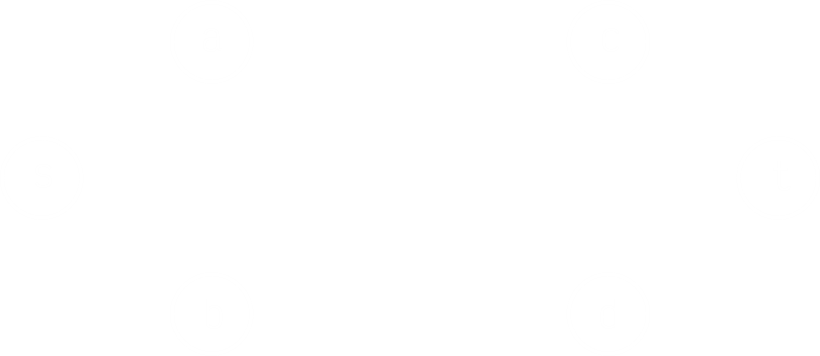
The reason we need to remember these two properties now is because they will help us find the required edges. Consider that we have an edge where and . Thus, . We do not have an edge , so . However, since , , meaning . This means we need to add an edge from to , which will have a weight of .

Example

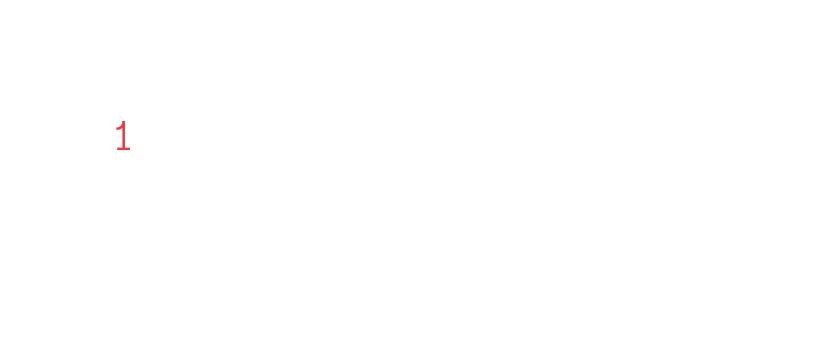


Say we have the above graph as our original network. From this graph, we will create our residual network.

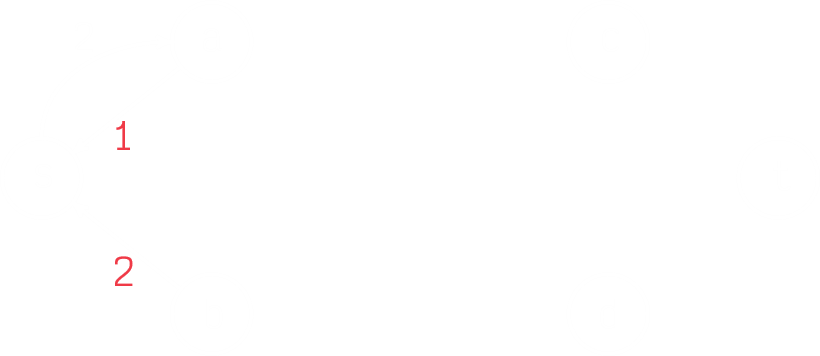
Initially, the network will be blank.



Consider the edge in the original network. and , so in the residual network, and we have an edge with weight . On the other hand, but , so and we need an edge with weight .

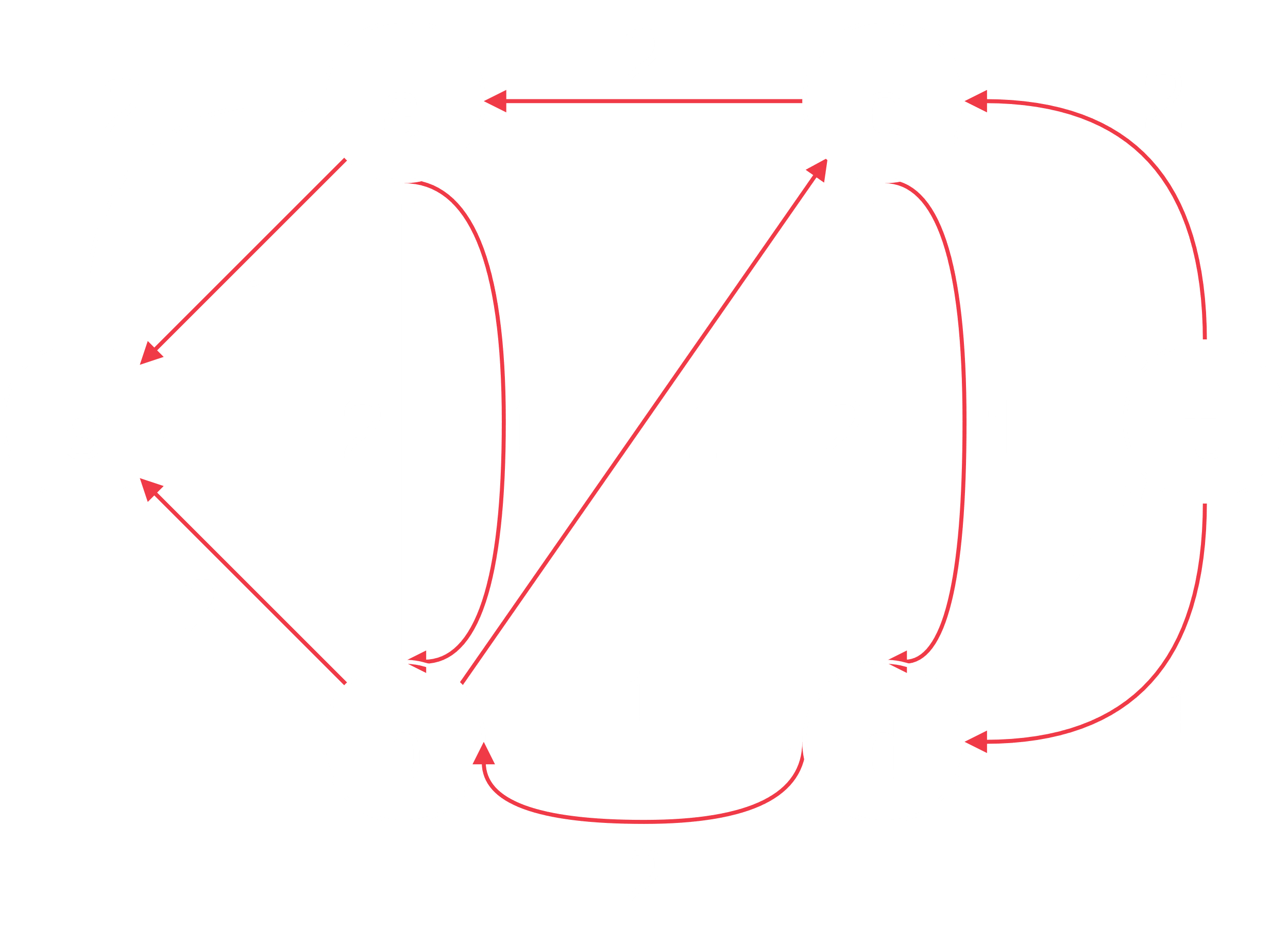


For the edge , and , so . On the reverse side, and , so . Since we can only consider positive weights, we have only one edge here.



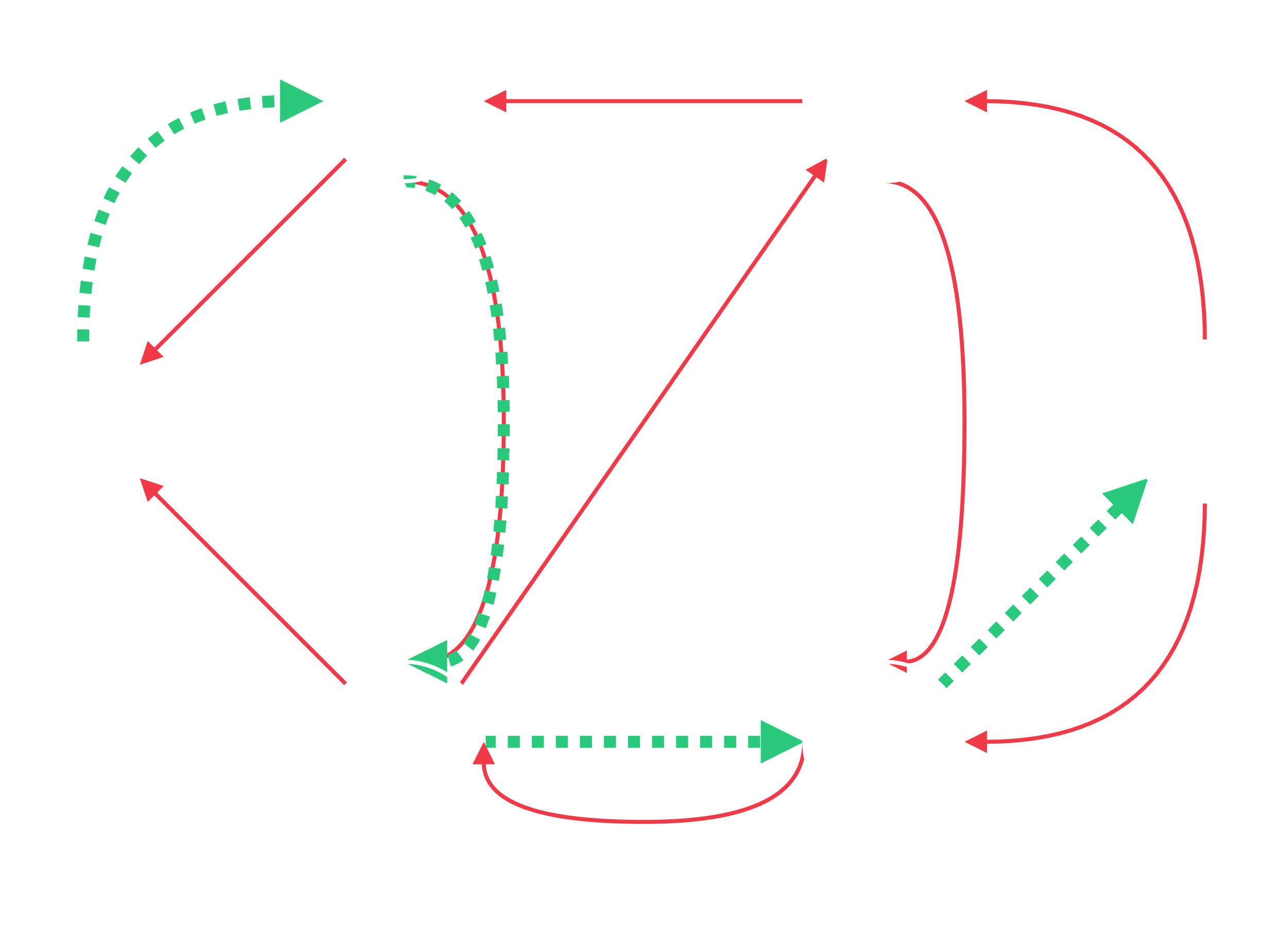
### Augmented Paths

Continuing in a similar manner, the final residual network we get will look like this:



Notice that the red edges are all edges created against the direction of the flow from the original graph.

The next thing we need to do is find a path from to . This can be any path. Say we choose this path:



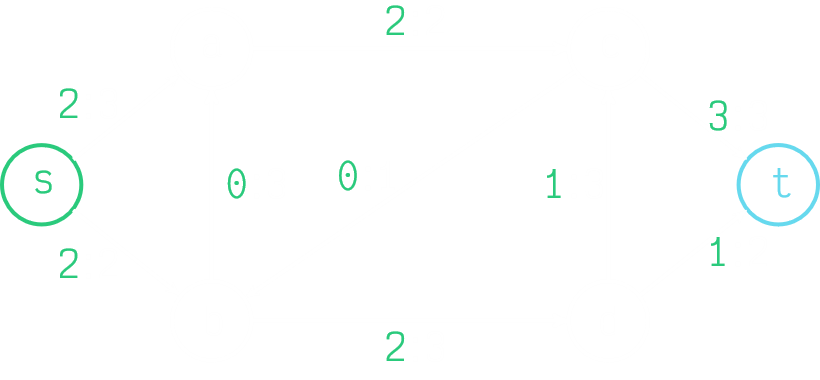
On this path, the black edges are ones where the flow should be increased and the red edges are ones where the flow should be decreased.

The path we just found is called the augmenting path. Following this path, if we make changes as suggested, the flow will be increased by the min value of the path. For the above path, the increase will be of . Formally, .

Using this method, we can find a solution to the problem.

1. From the original graph, we will create a residual graph, .
2. Then, we will find a path from to in .
3. Making the changes suggested by the path will cause our flow to increase by the min value of that path, so we will have a new graph.
4. From this new graph, we will again create a residual graph and repeat the process. We will keep repeating until we can no longer find a path in the residual graph.

Using this process, the final network we will get for the example we saw is this one:



Here, the flow value is .