Network Flow Review

CS 4820, Spring 2023

Sanjit Basker, Jenny Chen

Table of Contents

Table of Contents

Residual Graphs and Augmenting Paths

- Given a valid flow f we form the residual graph G_f

Residual Graphs and Augmenting Paths

- Given a valid flow f we form the residual graph G_f
- For an edge $e = (u, v)$ with capacity c_e, the residual graph contains a *forward edge* (u, v) of capacity $c_e - f(e)$ and a *backward edge* (v, u) of capacity $f(e)$

Residual Graphs and Augmenting Paths

- Given a valid flow f we form the residual graph G_f
- For an edge $e = (u, v)$ with capacity c_e, the residual graph contains a *forward edge* (u, v) of capacity $c_e - f(e)$ and a *backward edge* (v, u) of capacity $f(e)$
- Edges for which the capacities are 0 can be ignored

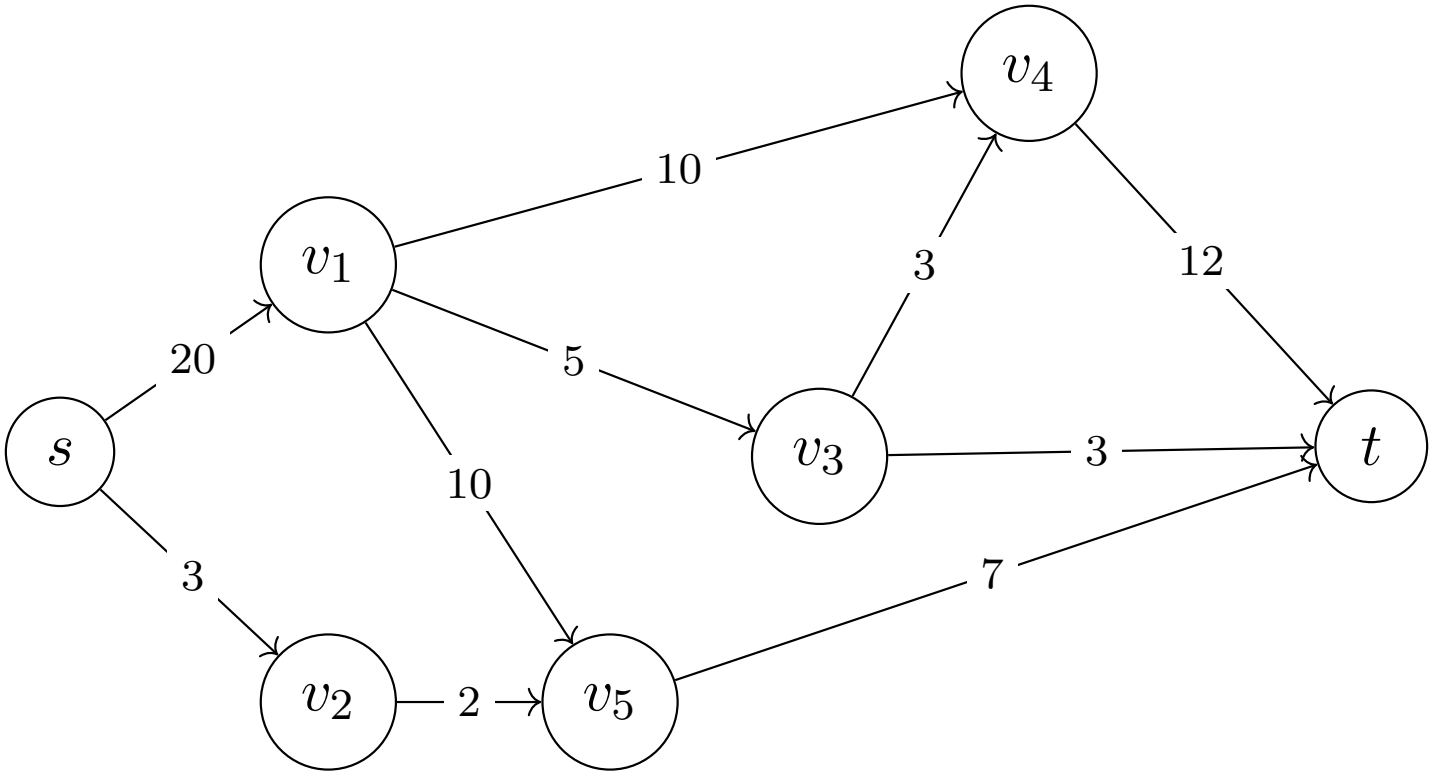
Residual Graphs and Augmenting Paths

- Given a valid flow f we form the residual graph G_f
- For an edge $e = (u, v)$ with capacity c_e, the residual graph contains a *forward edge* (u, v) of capacity $c_e - f(e)$ and a *backward edge* (v, u) of capacity $f(e)$
- Edges for which the capacities are 0 can be ignored
- An augmenting path in G_f is a path from s to t (on edges of positive residual capacity)

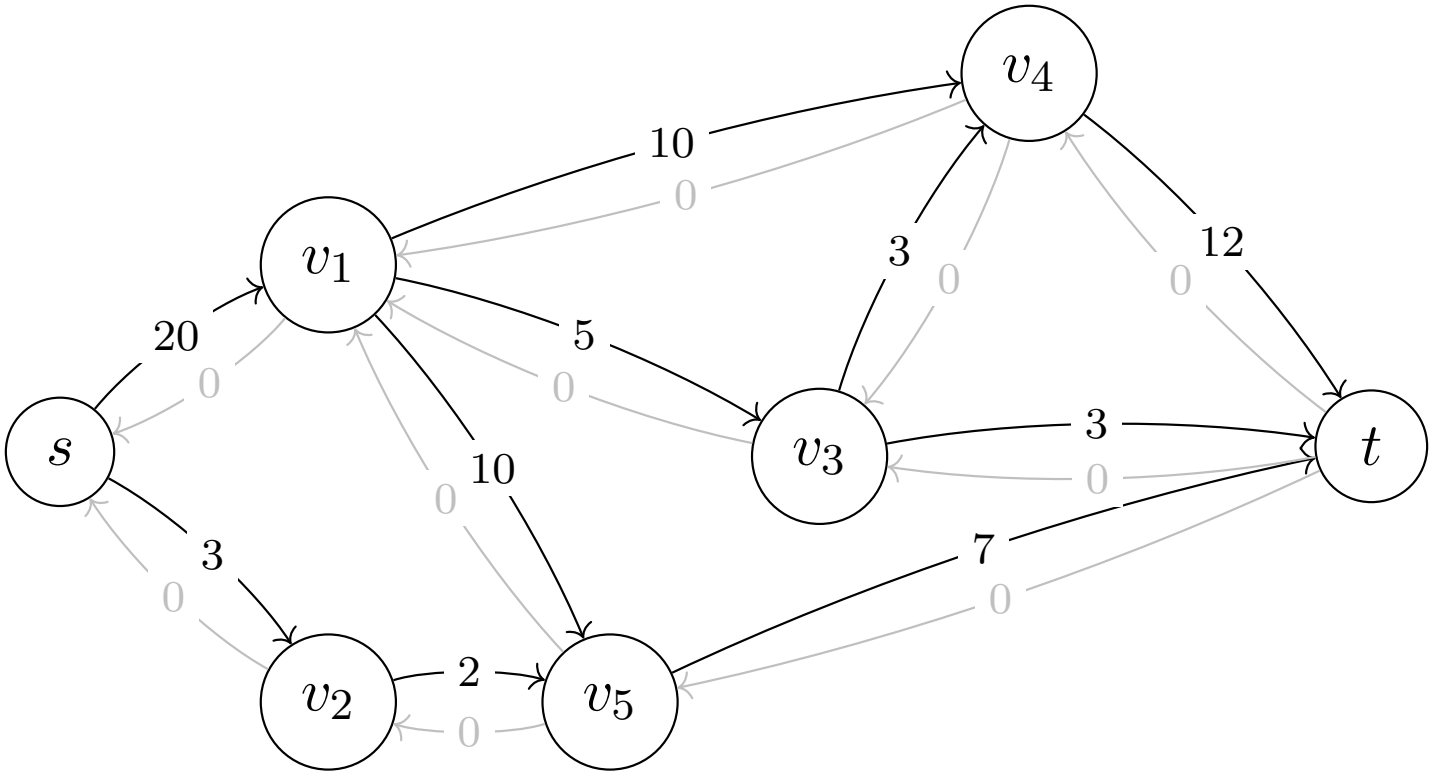
Residual Graphs and Augmenting Paths

- Given a valid flow f we form the residual graph G_f
- For an edge $e = (u, v)$ with capacity c_e, the residual graph contains a *forward edge* (u, v) of capacity $c_e - f(e)$ and a *backward edge* (v, u) of capacity $f(e)$
- Edges for which the capacities are 0 can be ignored
- An augmenting path in G_f is a path from s to t (on edges of positive residual capacity)
- If every edge on an augmenting path has capacity at least δ, we can "augment" f to get a new flow f^+, with $v(f^+) = v(f) + \delta$
  - When augmenting on an edge in the "forward" direction, we are pushing more flow
  - When augmenting on an edge in the "backward" direction we are undoing flow

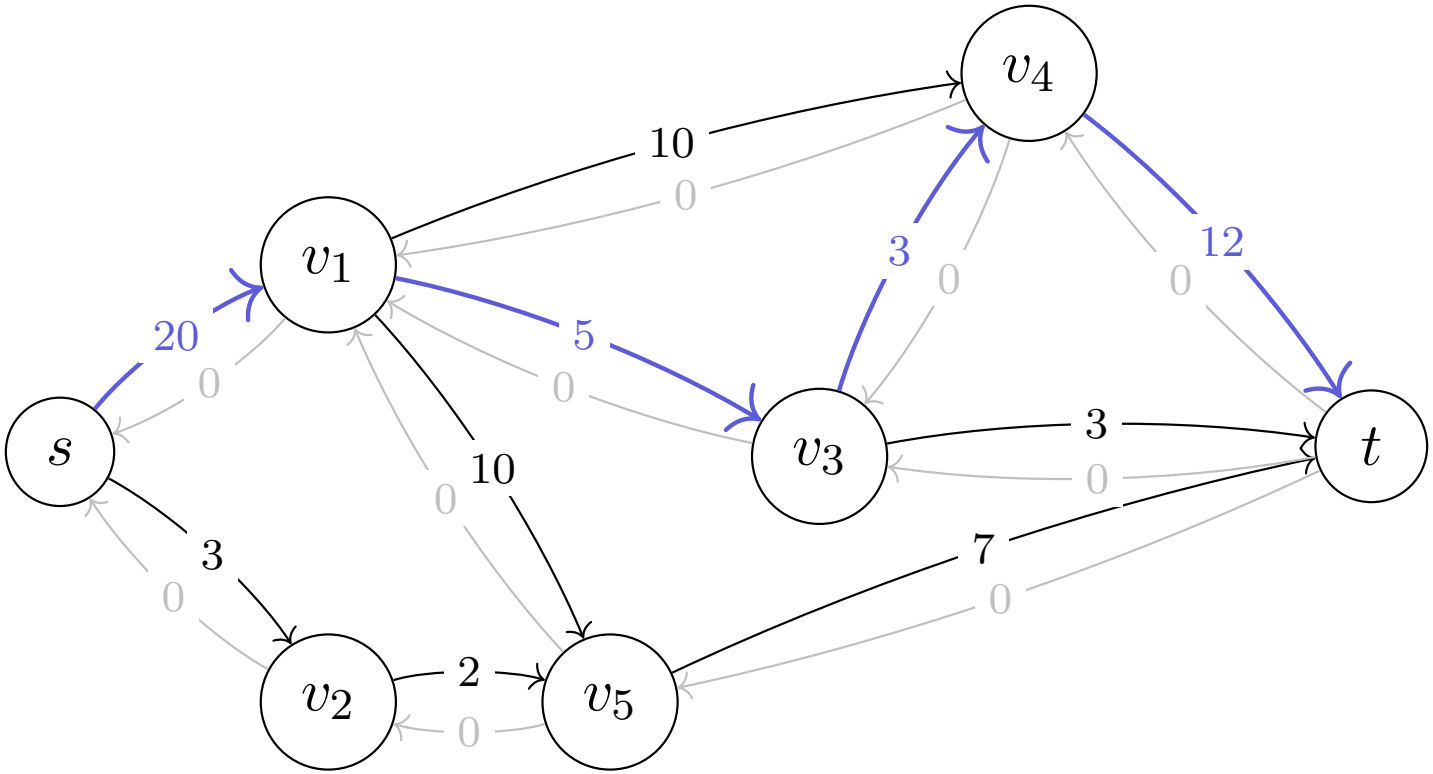
The Ford-Fulkerson Algorithm



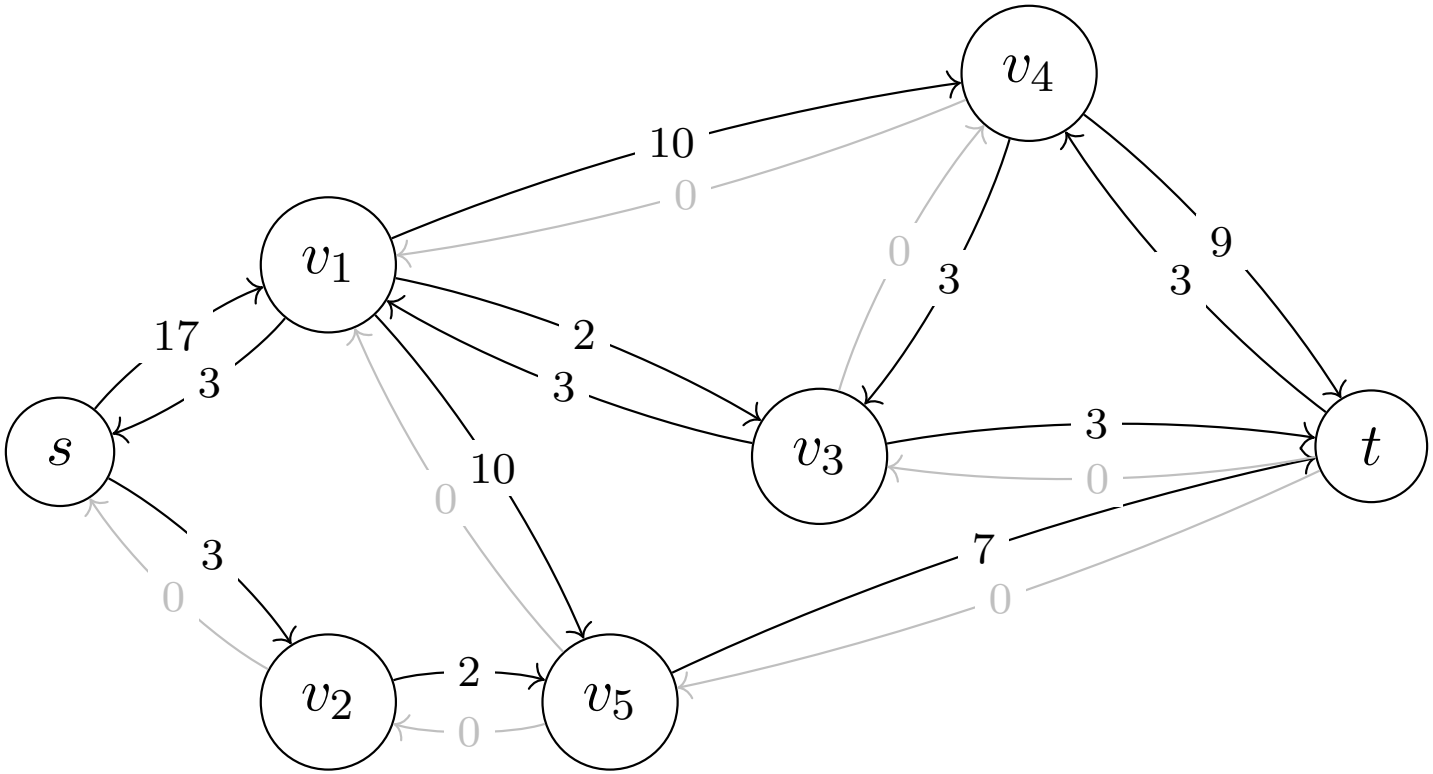
The Ford-Fulkerson Algorithm



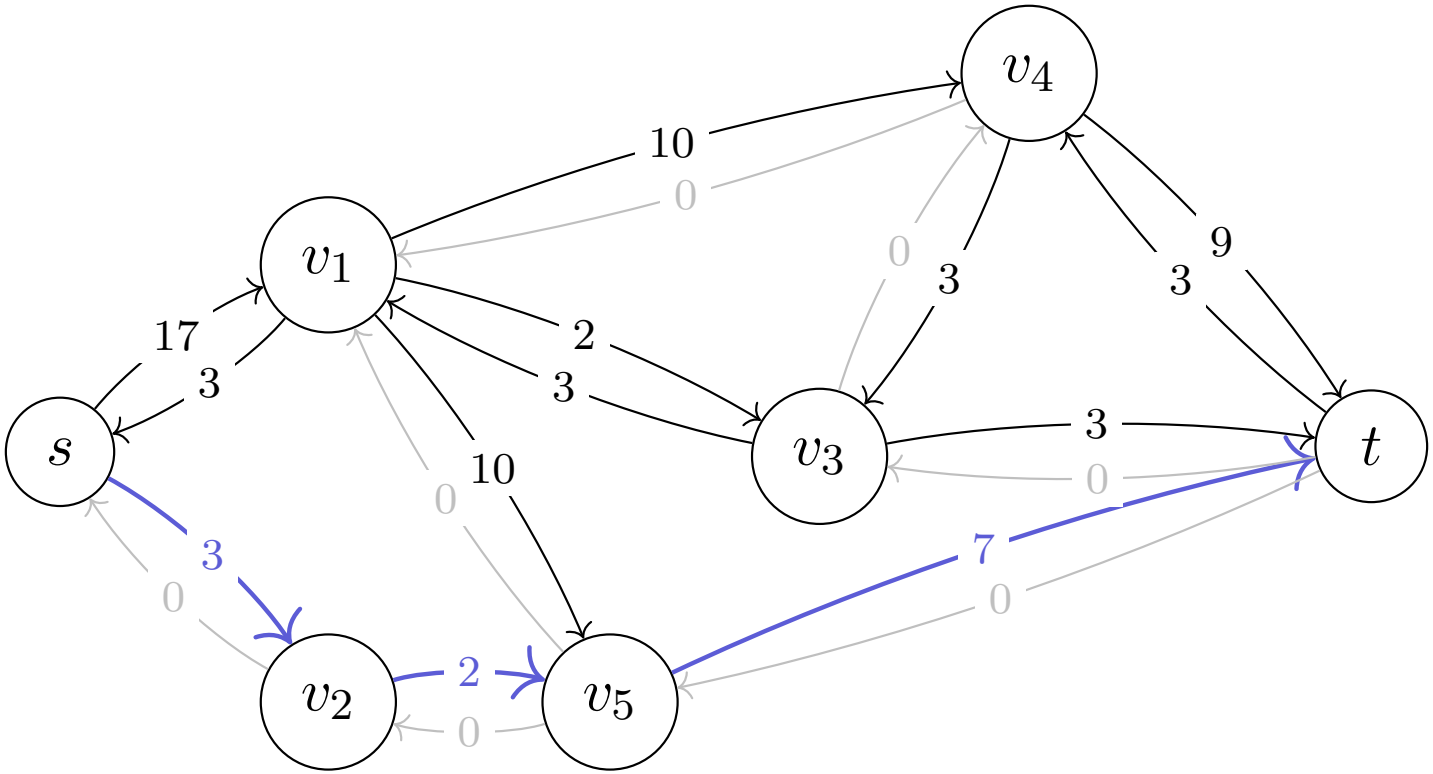
The Ford-Fulkerson Algorithm



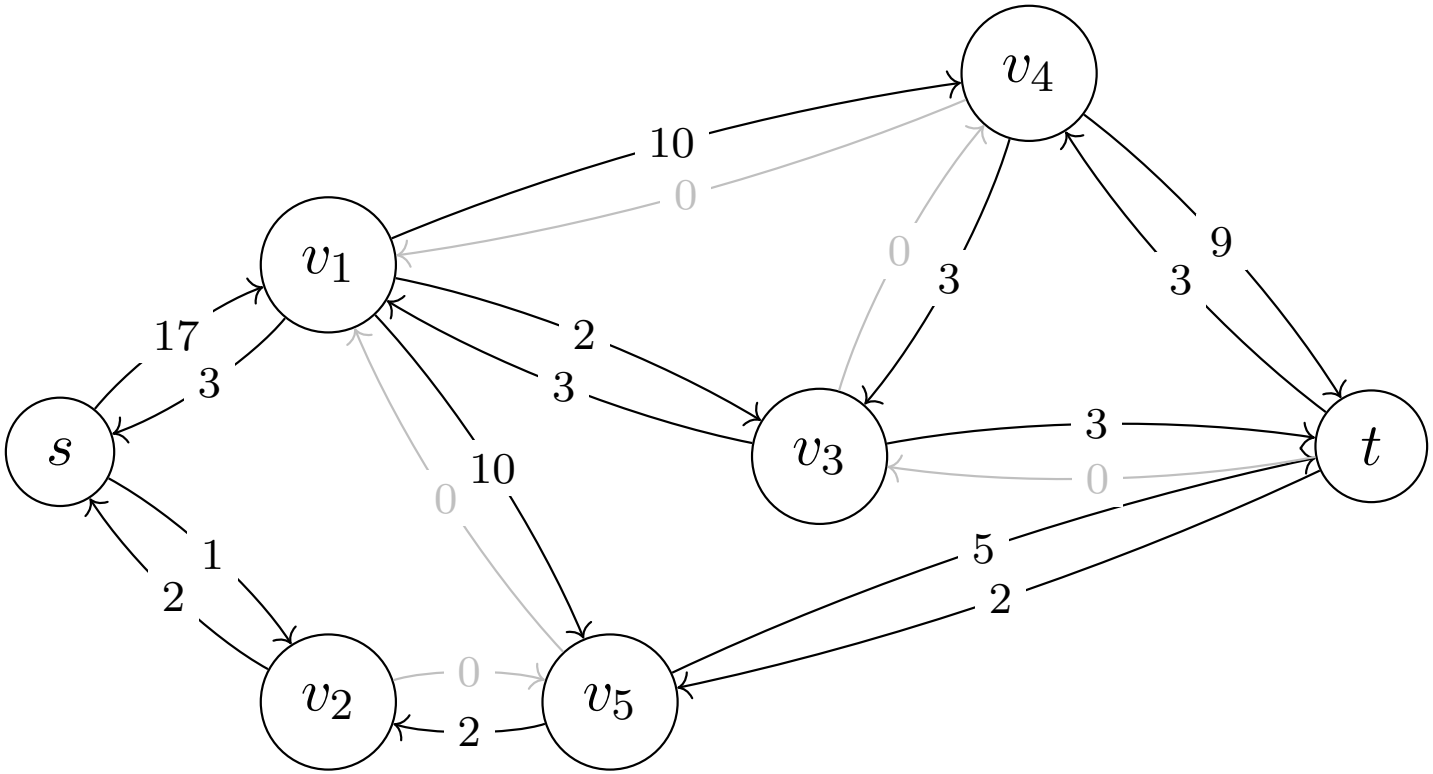
The Ford-Fulkerson Algorithm



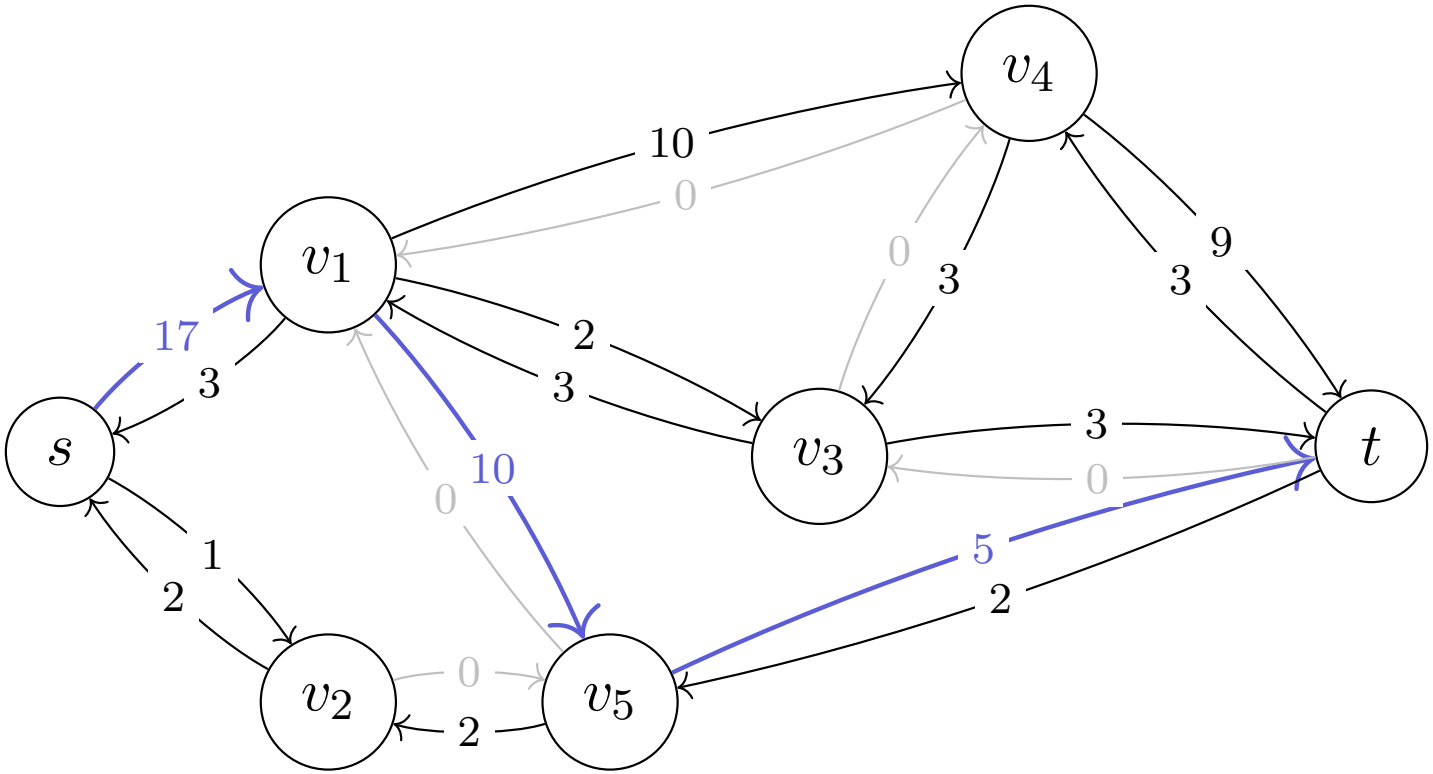
The Ford-Fulkerson Algorithm



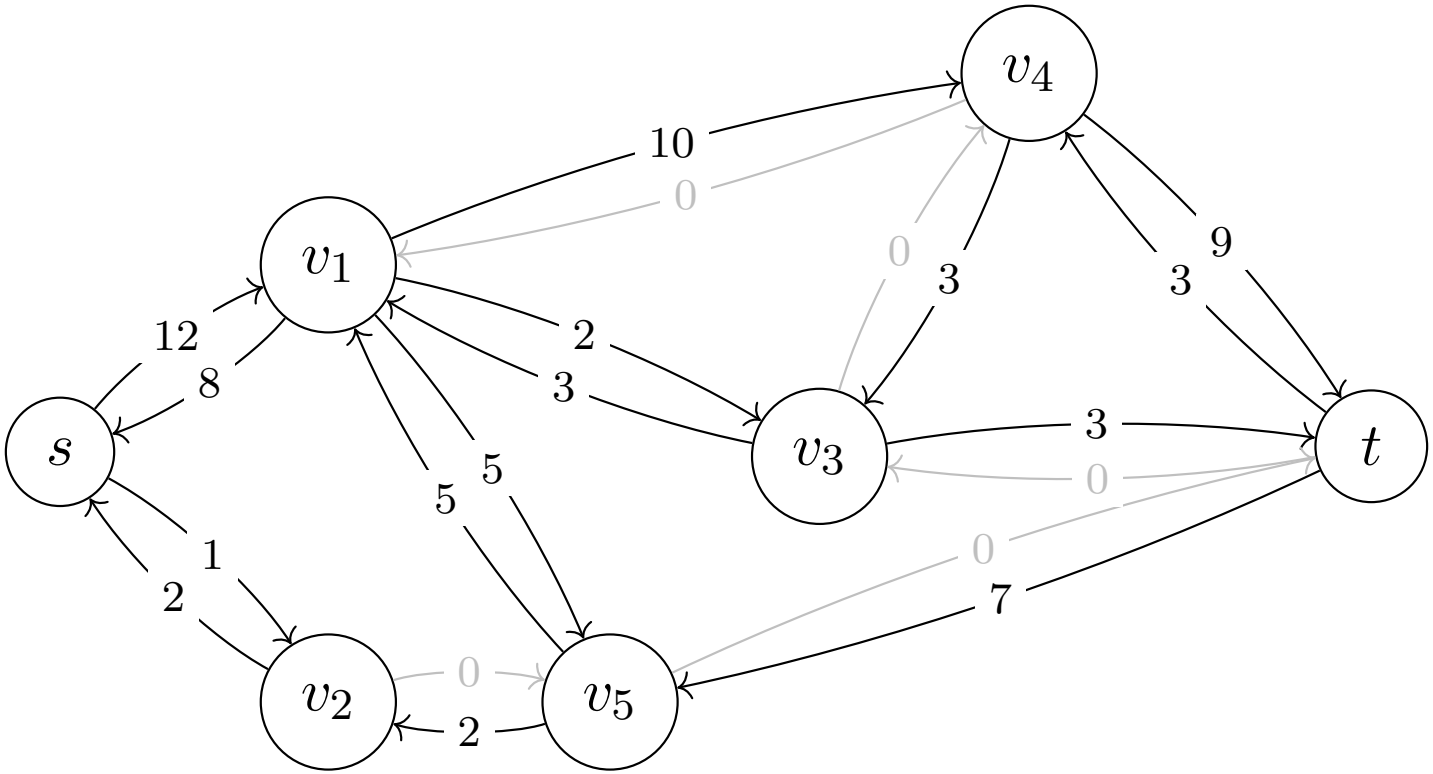
The Ford-Fulkerson Algorithm



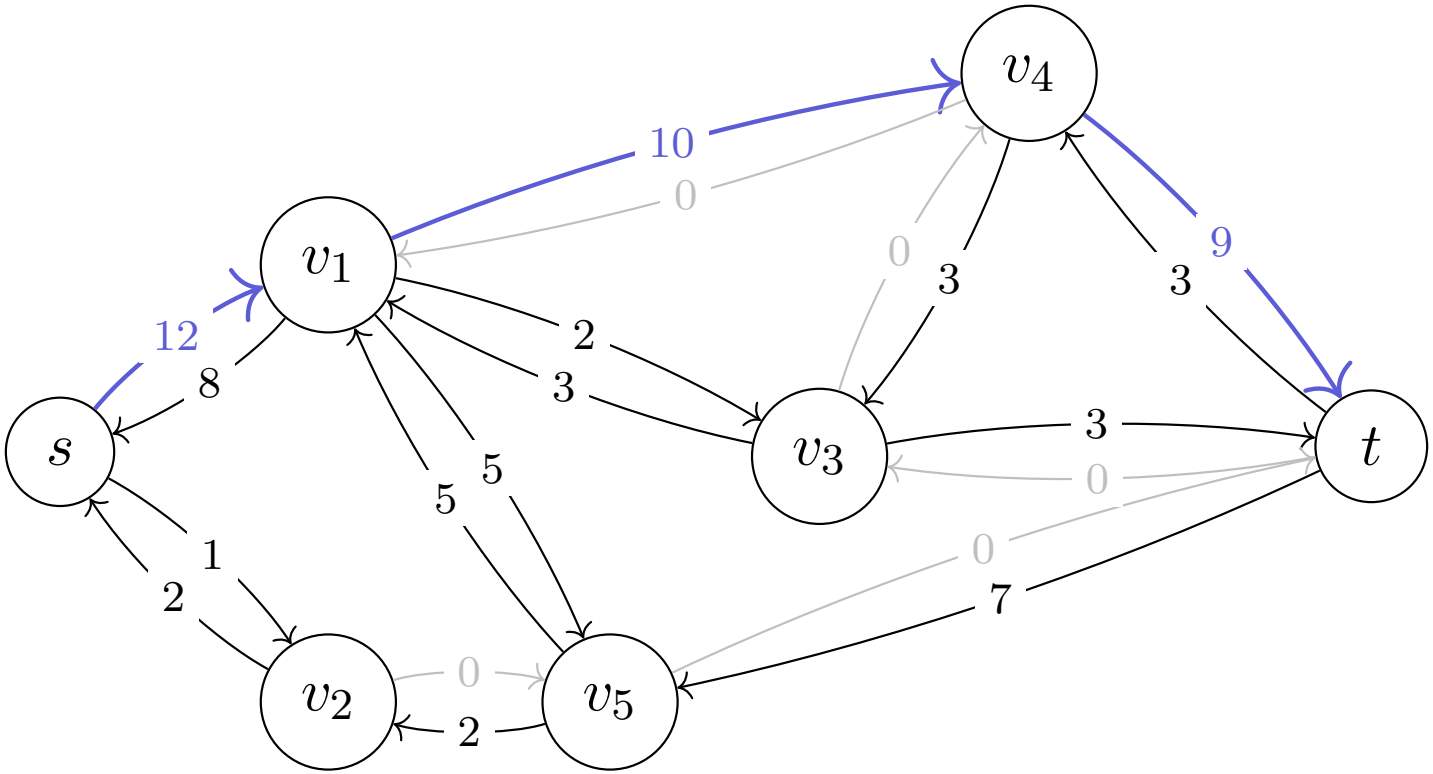
The Ford-Fulkerson Algorithm



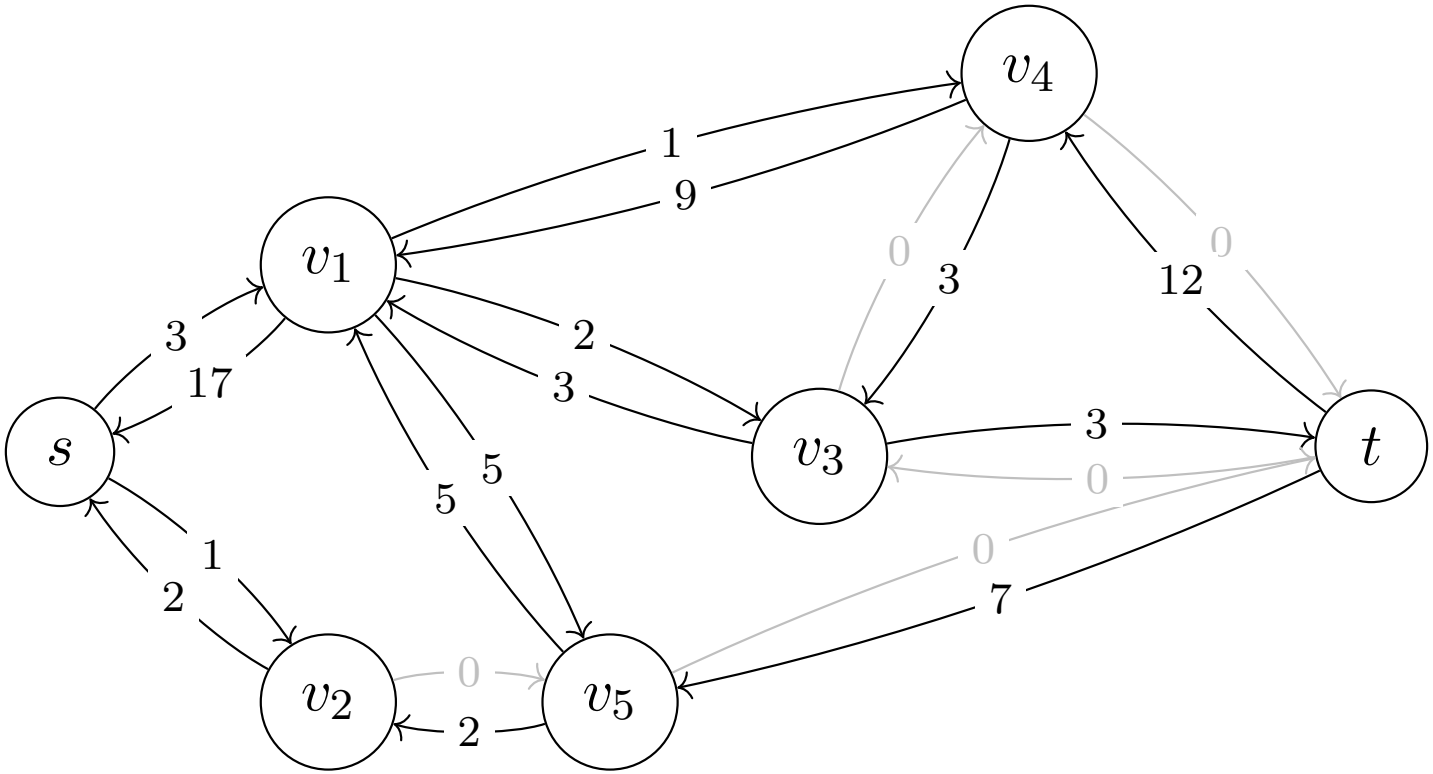
The Ford-Fulkerson Algorithm



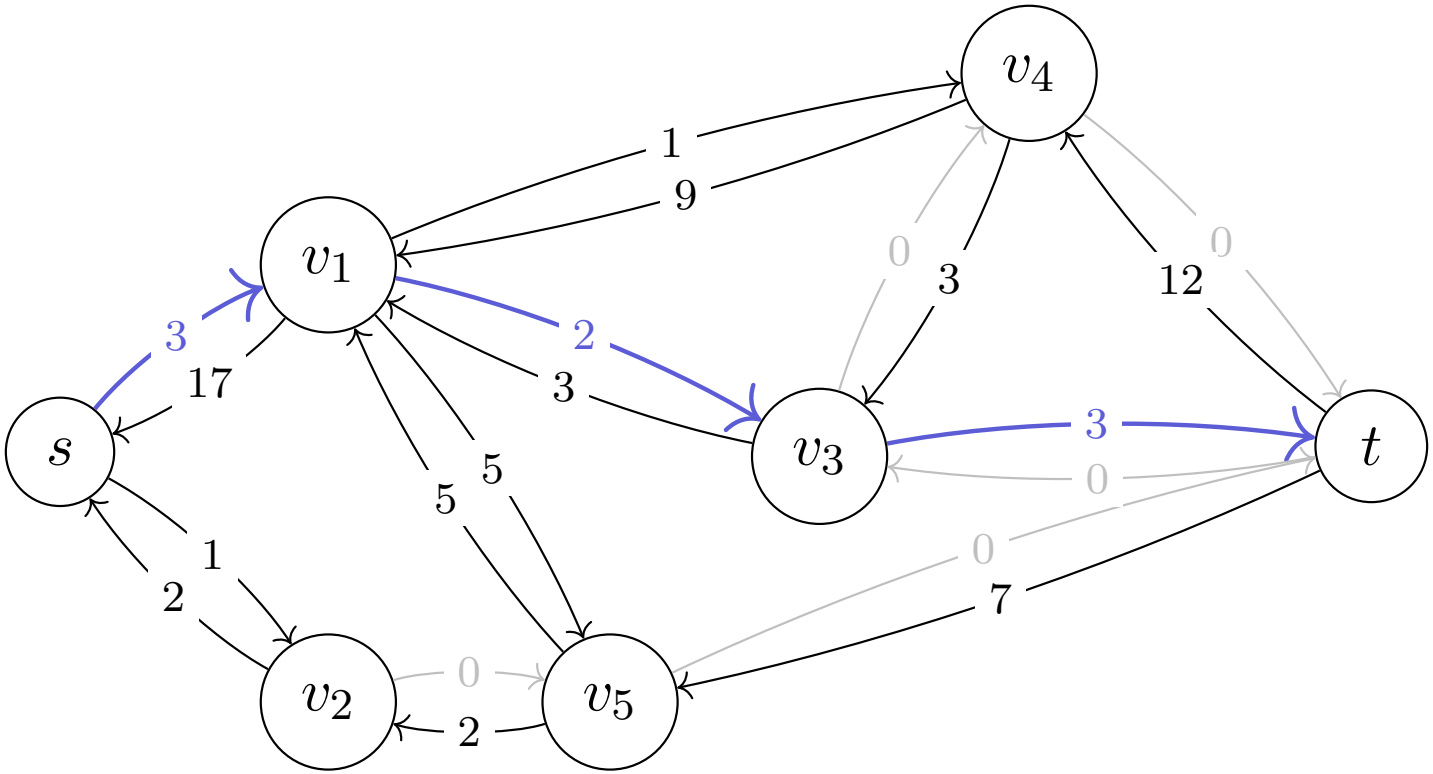
The Ford-Fulkerson Algorithm



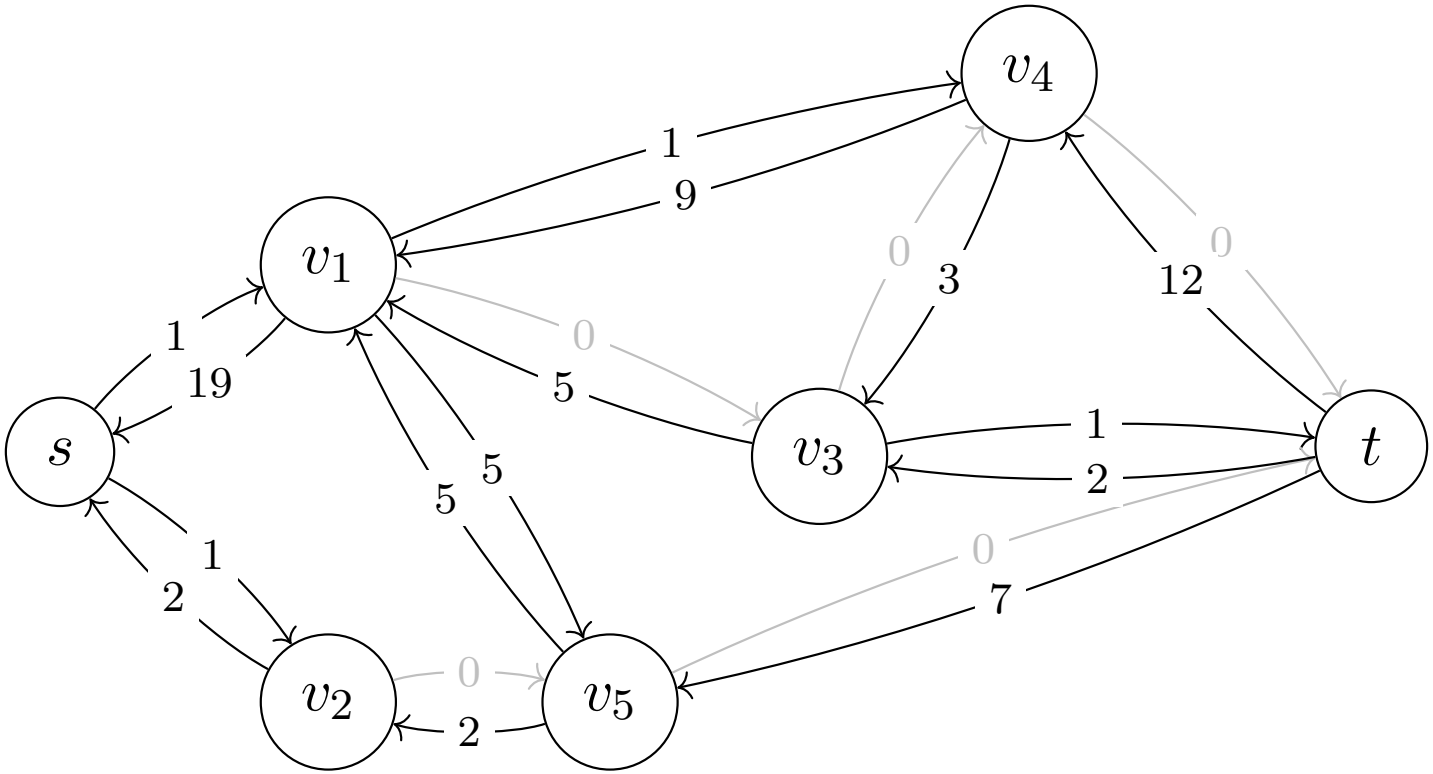
The Ford-Fulkerson Algorithm



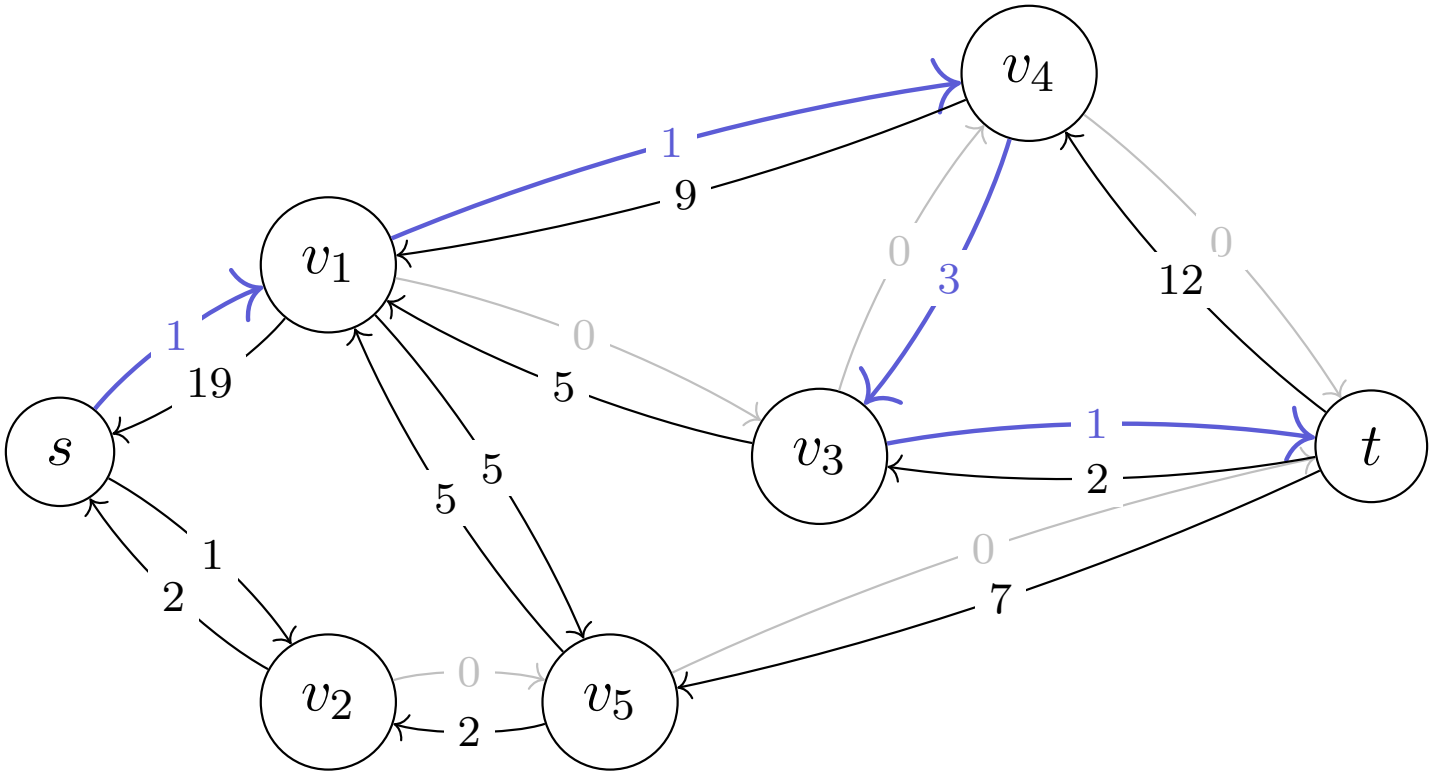
The Ford-Fulkerson Algorithm



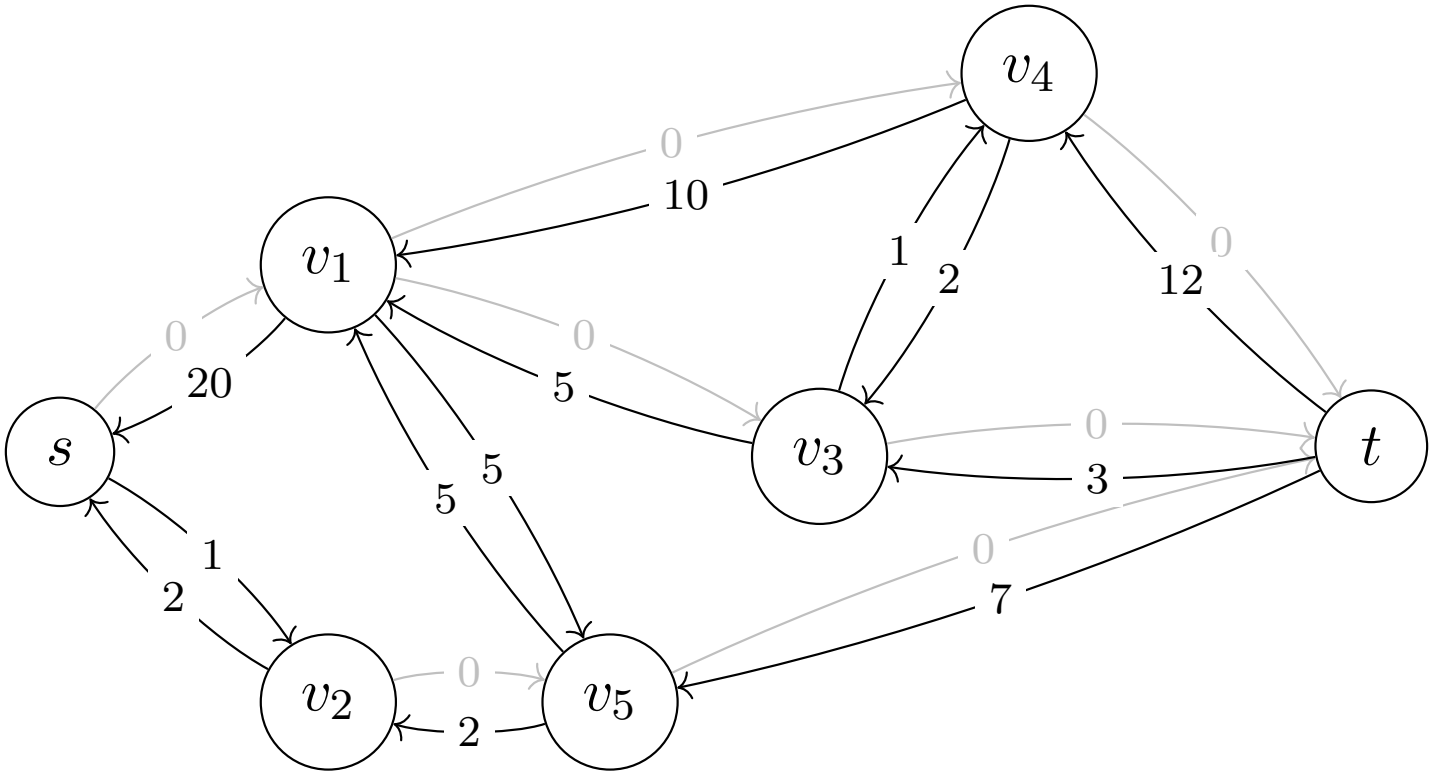
The Ford-Fulkerson Algorithm



The Ford-Fulkerson Algorithm

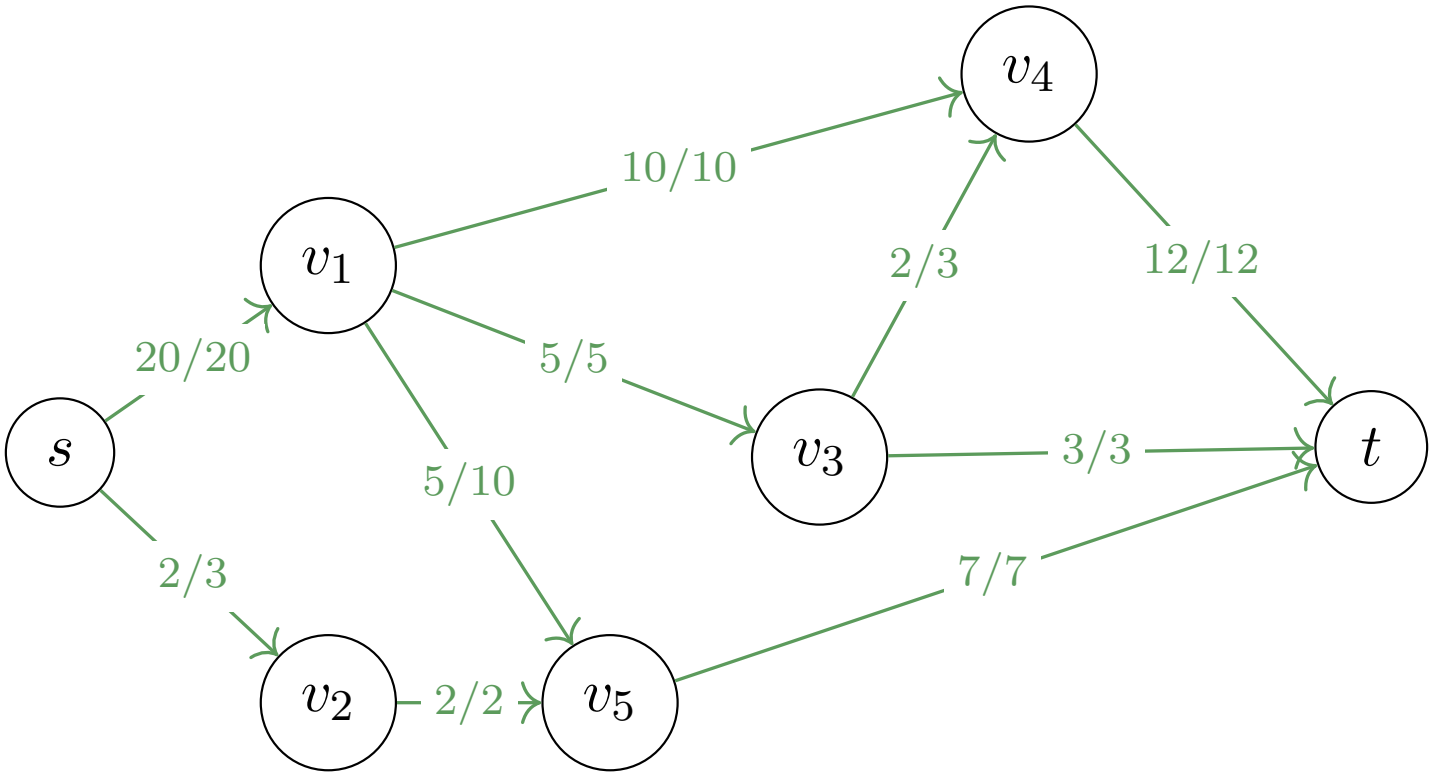


The Ford-Fulkerson Algorithm



Try doing a BFS from s on edges of positive residual capacity.

The Ford-Fulkerson Algorithm



All done!

Important Facts about Maximum Flows

Theorem (Max-Flow Min-Cut, 7.13)

In every flow network, the maximum value of an s-t flow is equal to the minimum capacity of an s-t cut.

Theorem (Flow Integrality, 7.14)

If all capacities in the flow network are integers, then there is a maximum flow f for which every flow value $f(e)$ is an integer.

Path Decomposition

Define the sum of two flows f and g by pointwise addition: it is a flow h where $h(e) = f(e) + g(e)$ for all edges e.
Define the *support* of a flow to be the set of edges that contain positive flow: $E(f) = \{e \in E : f(e) > 0\}$.

Lemma

Suppose f is a flow whose support is acyclic.
There exists a set $\{f_1, \ldots, f_k\}$ of flows whose sum is f and such that for every flow f_i in the set, the support of f_i is a s-t path. We can find such a set in $O(m^2)$ time.

Proof of Path Decomposition

The proof is by induction on the number of edges in $E(f)$.

- If $|E(f)| = 0$, we can output an empty decomposition and we are done.

Proof of Path Decomposition

The proof is by induction on the number of edges in $E(f)$.

- If $|E(f)| = 0$, we can output an empty decomposition and we are done.
- If $|E(f)| > 0$, suppose $e = (u_0, v_0)$ has positive flow on it. Repeatedly "follow the flow forwards" by looking for an edge (v_i, v_{i+1}) with positive flow. This process must terminate with $v_k = t$ because the number of edges is finite and support of the flow contains no cycles, meaning we'll never repeat an edge. Similarly, "follow the flow backwards" by looking for edges (u_{i+1}, u_i) until $u_{k'} = s$. The sequence of vertices $s, u_{k'-1}, \ldots, u_0, v_0, \ldots, u_{k-1}, t$ is an s-t path in the support of f.

Inductive Step, Continued

- Call the path P.
- Suppose the minimum flow on the edges in the path is δ, with $f(e^*) = \delta$. Write $f = f^- + g$, where

$$f^-(e) = \begin{cases} f(e) - \delta, & \text{if } e \in P \\ f(e), & \text{otherwise} \end{cases}, \quad g(e) = \begin{cases} \delta, & \text{if } e \in P \\ 0, & \text{otherwise} \end{cases}$$

- Note that if $f^-(e) > 0$, then $f(e) > 0$. Further, $f^-(e^*) = 0$. So f^- has a support strictly smaller than that of f.
- Find a path decomposition $f_1, f_2, \ldots, f_\ell$ of f^-, which we know exists by the inductive hypothesis.
- Set $f_{\ell+1} = g$ to get a path decomposition of f.

Table of Contents

Problem

- A hospital has already worked out its doctors' regular daily schedule. However, they still need to figure out the schedule for vacation days (Thanksgiving, Christmas, ...). They want to make sure that they have at least one doctor covering each vacation day.

Problem

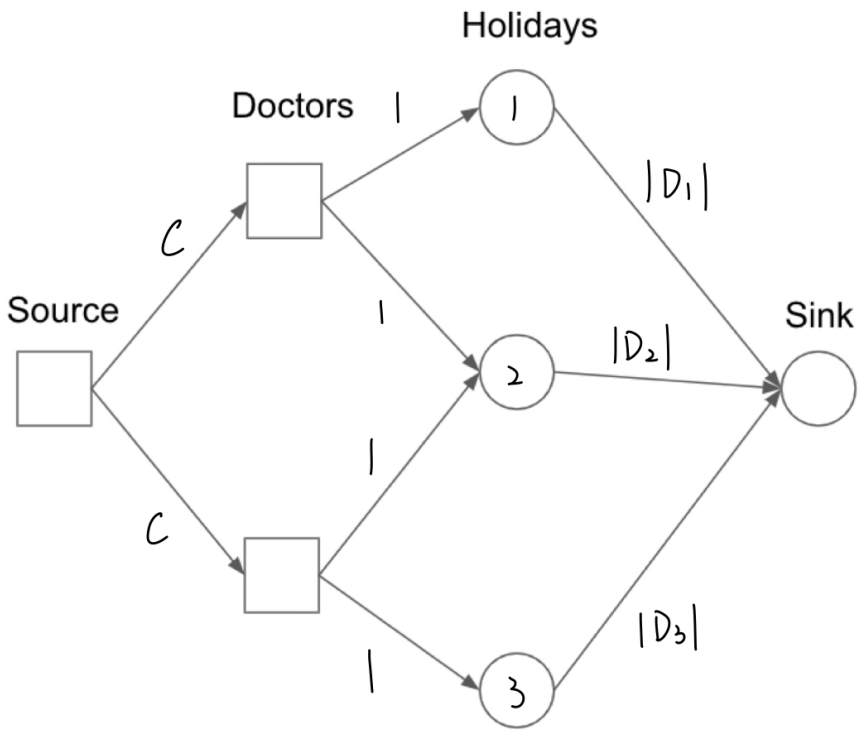
- A hospital has already worked out its doctors' regular daily schedule. However, they still need to figure out the schedule for vacation days (Thanksgiving, Christmas, ...). They want to make sure that they have at least one doctor covering each vacation day.
- There're k vacation periods, each spanning several contiguous days D_i.
- There're n doctors at the hospital, and each doctor i has a set of vacation days S_i when he or she is available to work.

Problem

- A hospital has already worked out its doctors' regular daily schedule. However, they still need to figure out the schedule for vacation days (Thanksgiving, Christmas, ...). They want to make sure that they have at least one doctor covering each vacation day.
- There're k vacation periods, each spanning several contiguous days D_i.
- There're n doctors at the hospital, and each doctor i has a set of vacation days S_i when he or she is available to work.
- We want a polynomial-time algorithm that determines whether it's possible to select a single doctor to work on each day under the following constraints:
  1. Given a parameter c, each doctor can only work for at most c vacation days in total (of course they need to be available).
  2. For each vacation period j, each doctor can only be assigned to at most one of the days in D_j.

First (Wrong) Attempt

Since this question is asking us to match the doctors with the vacation periods, with the constraints requiring each doctor cannot work for more than c days in total, it's natural to think about using network flow reduction to solve this question.
The first naive approach is simply having a column of doctors and a column of vacation periods.



First (Wrong) Attempt

First, let's make the construction clearer:

- To reinforce constraint 1, all edges from the source to doctors have capacity c.

First (Wrong) Attempt

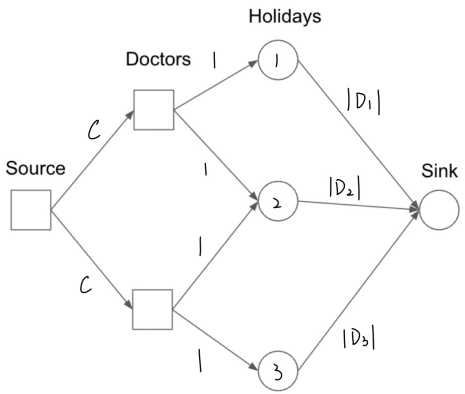
First, let's make the construction clearer:

- To reinforce constraint 1, all edges from the source to doctors have capacity c.
- To ensure we only assign available doctors to the vacation period, we only connect doctors to the holiday node when they can work during that time (i.e. $S_i \cap D_j \neq \varnothing$). The capacity of those edges will be 1 to make sure they're only assigned once in each holiday.

First (Wrong) Attempt

First, let's make the construction clearer:

- To reinforce constraint 1, all edges from the source to doctors have capacity c.
- To ensure we only assign available doctors to the vacation period, we only connect doctors to the holiday node when they can work during that time (i.e. $S_i \cap D_j \neq \varnothing$). The capacity of those edges will be 1 to make sure they're only assigned once in each holiday.
- Because we want at least one doctor covering each vacation day, the capacity for edges from holidays to the sink should be $|D_i|$, the number of days the vacation period has.



Convert Flow to Output

Remember, we're using flow reduction, meaning we're using the FF algorithm to help solve this doctor question. Your algorithm might look something like this:

1. Prepare the input as described
2. Run FF
3. Convert output back to this problem

Convert Flow to Output

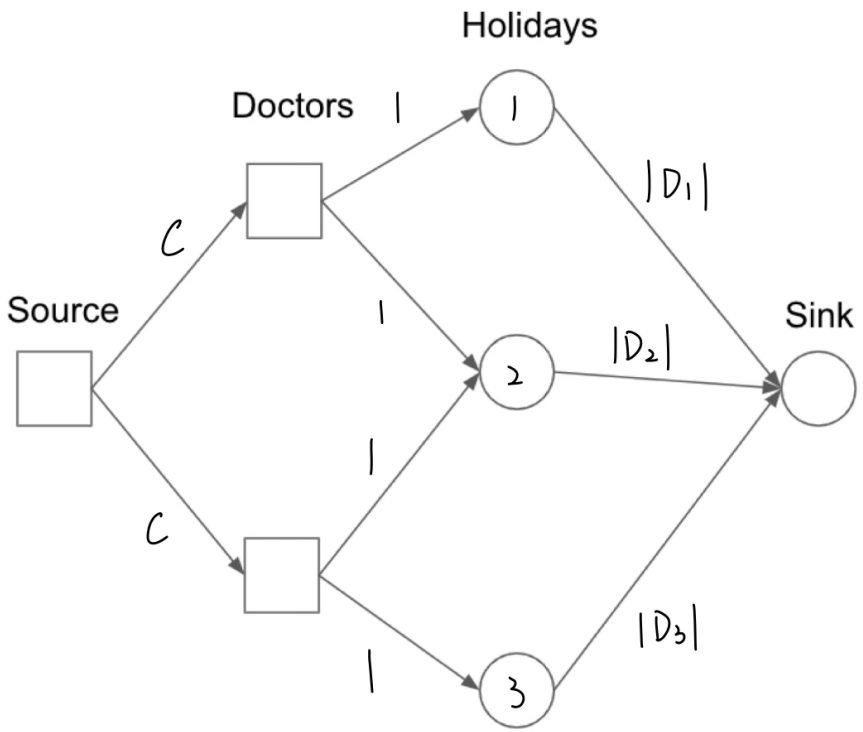
Remember, we're using flow reduction, meaning we're using the FF algorithm to help solve this doctor question. Your algorithm might look something like this:

1. Prepare the input as described
2. Run FF
3. Convert output back to this problem

If the flow has a value of $D = |\bigcup_j D_j|$, then we say there is a valid assignment. Furthermore, we can read out the assignment from the flow since if there's a flow from one doctor to a holiday, it means that we assign that doctor to that period of time. If the max flow is smaller than D, then we say there's no possible assignment (not all days are covered).

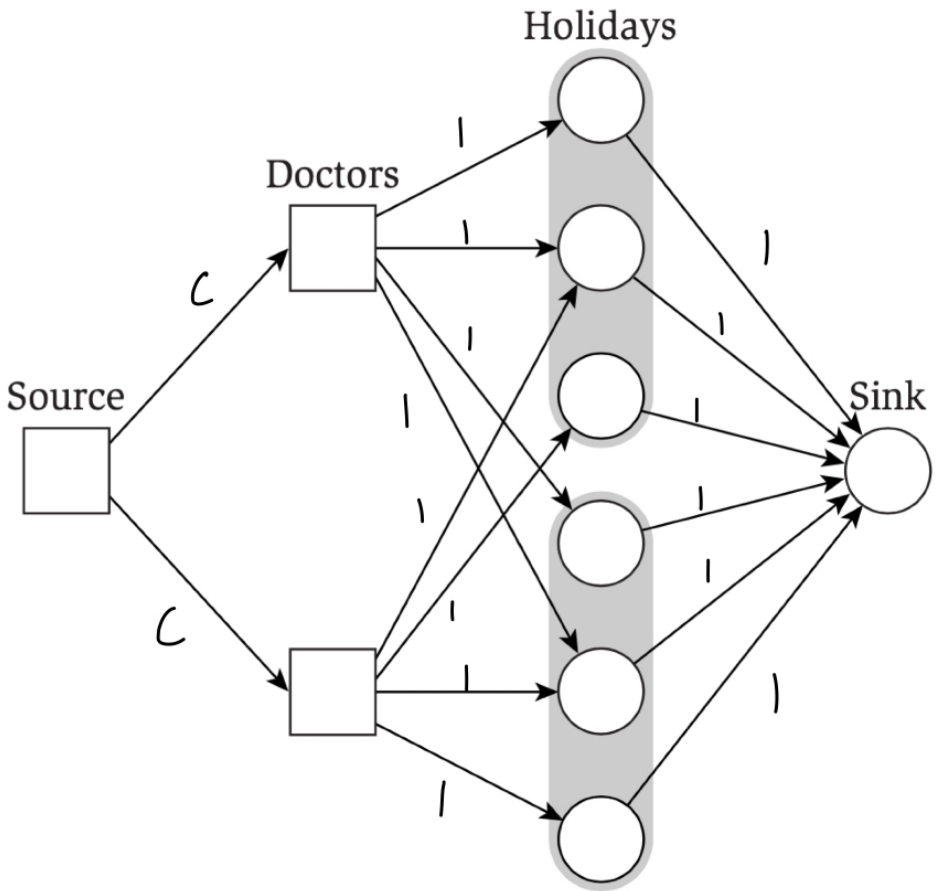
What's Missing

- Apparently, we didn't use all the information that the problem provided us, which is not a good sign.
- We don't know which day the doctors are assigned to (maybe they're assigned to the wrong days as well) since all we get is whether enough doctors will be assigned to each holiday period.



A Slightly Better Construction

Now we realize that we need to treat each day of the holiday differently. The modified graph will look like this:

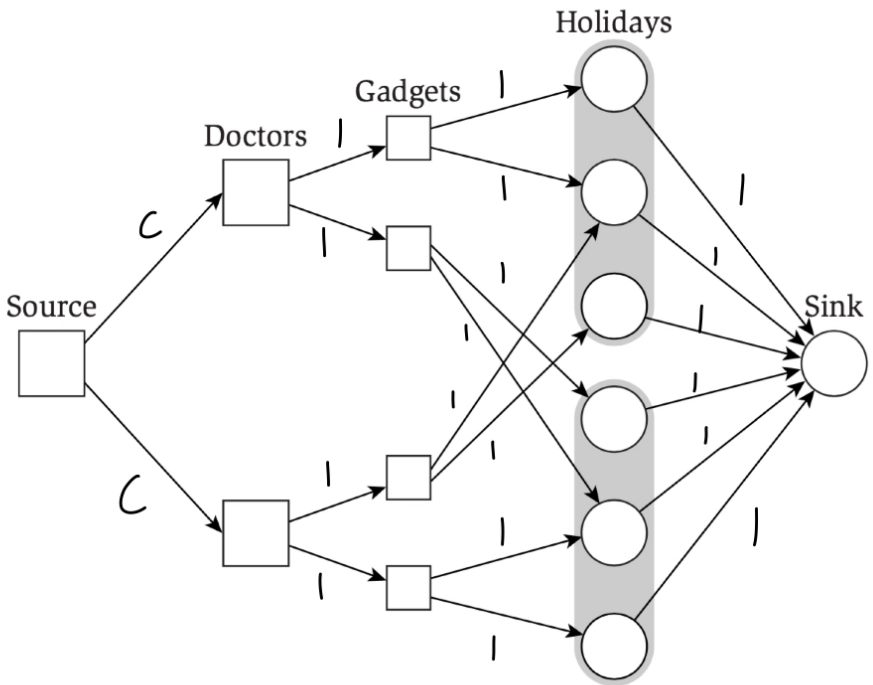


A Slightly Better Construction

Like the previous construction, all (source, doctor) edges have capacity c. We only connect doctor d_i to holiday h_j if the doctor is available. These edges have a capacity of 1 since one flow means we assign one doctor to that day. All (holiday, sink) edges have capacity 1. If at the end of FF the max flow equals to $D = |\bigcup_j D_j|$, then we know all days in the vacation period are covered.

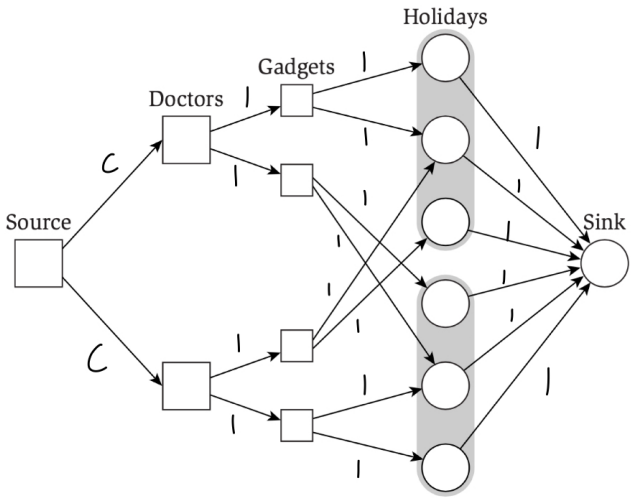
This solved the previous problem, now we know which day the doctors are assigned to. However, we cannot satisfy constraint 2 since there's nothing stopping the FF to push several flows through one doctor to several days in one vacation period.

The Gadgets



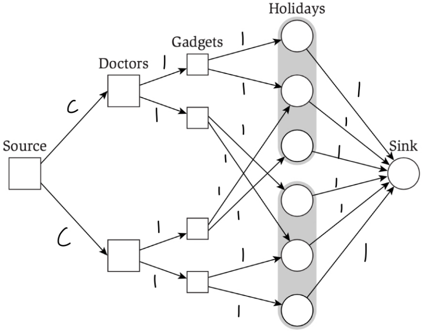
We create k "holiday gadgets" for each doctor, each corresponding to one holiday. You can see in the graph, each doctor has two gadgets since there're two holidays.

The Gadgets



The edges from source to doctors and from holidays to the sink remain the same. Let g_{ij} represent the holiday gadget for doctor d_i's availability for holiday h_j. We add edge (d_i, g_{ij}) if $S_i \cap D_j \neq \varnothing$; the edge has capacity 1. Say h_{jk} is the kth day of h_j. We add edge (g_{ij}, h_{jk}) for all $h_{jk} \in S_i$. They have capacities 1, just as the (doctor, holiday) edges in the previous construction.

The Gadgets



Since all (d_i, g_{ij}) edges have capacity 1, each doctor now can only get assigned to holiday once, which satisfies constraint 1.

Proof of Correctness

We'll briefly restate the algorithm. First, given all the information about the doctors and vacation periods, construct the graph in the previously discussed way. Then we run FF and get the max flow. If the max flow value equals $|\bigcup_j D_j|$, then output Yes; otherwise, output No. If the algorithm outputs Yes, the assignment is the following: we assign doctor d_i to h_{jk} if there's a flow from d_i to g_j and g_j to h_{jk}.
To prove our algorithm is correct, we need to show that there is a way to assign doctors to vacation days in a way that respects all constraints if and only if the max flow value is $D = |\bigcup_j D_j|$ in the flow network we constructed.

Proof of Correctness, First Direction

Suppose there's a valid way to assign doctors that satisfies all the constraints, then we can construct a valid flow with flow value D.

Proof of Correctness, First Direction

Suppose there's a valid way to assign doctors that satisfies all the constraints, then we can construct a valid flow with flow value D.
If doctor d_i is assigned to holiday h_{jk}, then we push one flow along source $\rightarrow d_i \rightarrow g_j \rightarrow h_{jk} \rightarrow$ sink.

Proof of Correctness, First Direction

Suppose there's a valid way to assign doctors that satisfies all the constraints, then we can construct a valid flow with flow value D.
If doctor d_i is assigned to holiday h_{jk}, then we push one flow along source $\rightarrow d_i \rightarrow g_j \rightarrow h_{jk} \rightarrow$ sink.
Now we both need to prove this is a valid flow (K&T pg339), and it's flow value is D.

Proof of Correctness, First Direction

Suppose there's a valid way to assign doctors that satisfies all the constraints, then we can construct a valid flow with flow value D.
If doctor d_i is assigned to holiday h_{jk}, then we push one flow along source $\rightarrow d_i \rightarrow g_j \rightarrow h_{jk} \rightarrow$ sink.
Now we both need to prove this is a valid flow (K&T pg339), and it's flow value is D.

- Capacity condition: For each edge e, $0 \leq f(e) \leq c_e$
  - Normally this directly follows from the construction. All capacities are satisfied since, each doctor is assigned at most c times, and only get assigned at most once for each holiday.

Proof of Correctness, First Direction

Suppose there's a valid way to assign doctors that satisfies all the constraints, then we can construct a valid flow with flow value D.
If doctor d_i is assigned to holiday h_{jk}, then we push one flow along source $\rightarrow d_i \rightarrow g_j \rightarrow h_{jk} \rightarrow$ sink.
Now we both need to prove this is a valid flow (K&T pg339), and it's flow value is D.

- Capacity condition: For each edge e, $0 \leq f(e) \leq c_e$
  - Normally this directly follows from the construction. All capacities are satisfied since, each doctor is assigned at most c times, and only get assigned at most once for each holiday.
- Conservation condition: For each node v other than source and sink, flow in $=$ flow out: $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$
  - Because each time we send a flow from source to sink, this automatically satisfied (no flow is trapped).

Proof of Correctness, First Direction

Suppose there's a valid way to assign doctors that satisfies all the constraints, then we can construct a valid flow with flow value D.
If doctor d_i is assigned to holiday h_{jk}, then we push one flow along source $\rightarrow d_i \rightarrow g_j \rightarrow h_{jk} \rightarrow$ sink.
Now we both need to prove this is a valid flow (K&T pg339), and it's flow value is D.

- Capacity condition: For each edge e, $0 \leq f(e) \leq c_e$
  - Normally this directly follows from the construction. All capacities are satisfied since, each doctor is assigned at most c times, and only get assigned at most once for each holiday.
- Conservation condition: For each node v other than source and sink, flow in $=$ flow out: $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$
  - Because each time we send a flow from source to sink, this automatically satisfied (no flow is trapped).
- The flow value will be D since each time we assign a doctor to one day, we add the flow by one. Since the assignment is valid, it means all D days are covered, hence the total flow value is also D.

Proof of Correctness, Second Direction

Now we prove if there's a flow with flow value D, then we can find a valid doctor assignment that satisfies all constraints. Note that our graph contains no cycles. In particular, each path from s to t is of the form source $\rightarrow d_i \rightarrow g_j \rightarrow h_{jk} \rightarrow$ sink and thus ends in an edge of capacity 1.
So a flow with value D can be decomposed into D path flows.
For each path flow in the decomposition, assign doctor d_i to h_{jk}.
It's a valid assignment because:

- (source, doctor) edges have capacity c, so each doctor will be assigned for at most c days, satisfies constraint 1.
- Using the gadget, each doctor can only push one flow to one holiday, satisfies constraint 2.
- There's a total of D flow, meaning all D days are covered.

You can follow the same proof structure in your HW.

Runtime Analysis

The runtime consists of several parts:

1. Pre-process/Preparing the graph
   - This step is basically counting the number of nodes and edges you added to the graph. For this problem, we have $2 + n + nk + D$ nodes, where $D = |\bigcup_j D_j|$. We have at most $m = n + nk + nD + D$ edges if each doctor is available for all days. So it's $O(nD)$.
2. Running FF
   - This step normally dominates the runtime. Because we're assuming integer flows, and we know each time the flow is increased by at least one, the number of iterations is bounded by D. So it takes $O(mD) = O((n + nk + nD + D)D) = O(nD^2)$.
3. Converting output
   - To read the assignment from the flow, we iterate through some subset of the edges. So this step is $O(m) = O(nD)$.

The total runtime is therefore $O(nD^2)$