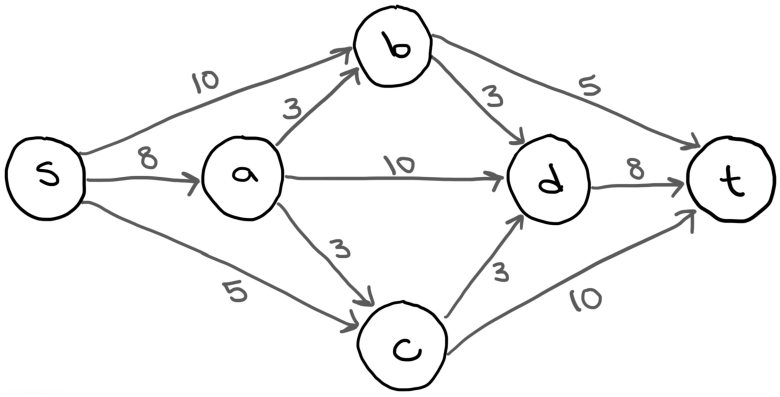
ADS Tutorial 5 Solutions

Instructor: Aris Filos-Ratsikas Tutor: Kat Molinet

November 4, 2025

Problem 1

Consider the following flow network, with edge capacities as indicated in the figure.



Apply the Ford-Fulkerson algorithm to this flow network to compute the value of the maximum flow $f_{\max}$ from s to t. In every step of the algorithm draw the residual graph, and indicate the value of the current flow, the augmenting path chosen, and the bottleneck capacity of that path.

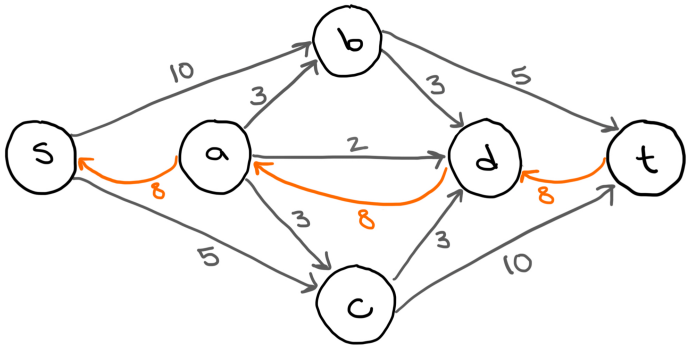
In the flow network above, indicate a minimum cut (as two sets of nodes S and T with $s \in S$ and $t \in T$). Provide an argument for why the cut you have indicated is minimal.

Solution

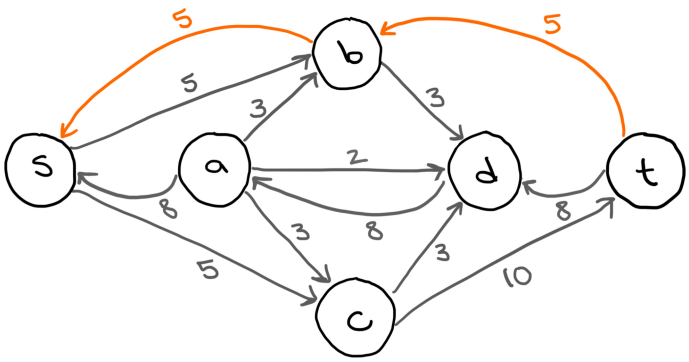
Note: Intermediate steps may vary, depending on which path is selected at each step.

To start, we choose to send flow along path $s - a - d - t$. The exact path we choose here doesn't matter, as long as it is able to carry flow from s to t (i.e., the edges of the path have sufficient capacity). Given the edge capacities, the maximum amount of flow we can send along this path (a.k.a., the bottleneck capacity) is 8. Thus, the total amount of flow in the network increases from 0 to 8. Below, we update the residual graph to reflect the 8 units of flow we've sent through the network. In addition to decreasing the capacities of the edges that carried flow in this step, we also add a backwards-facing edge for each step of the path. These backwards edges will allow us to effectively "undo" some portion of the flow we just sent through the network, if it later turns out that our choice to send 8 units of flow through $s - a - d - t$ was not optimal. In summary:

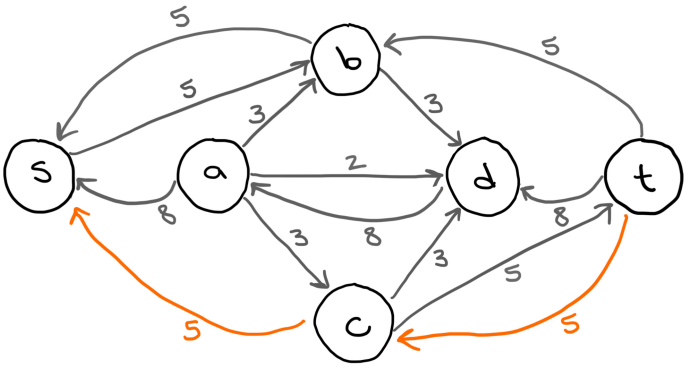
Augmenting path chosen: $s - a - d - t$; bottleneck capacity: 8; $f_1 : 0 \to 8$; residual graph:



We preceed as in the previous step, this time choosing to send 5 units of flow along path $s - b - t$.
Augmenting path chosen: $s - b - t$; bottleneck capacity: 5; $f_2 : 8 \to 13$; residual graph:

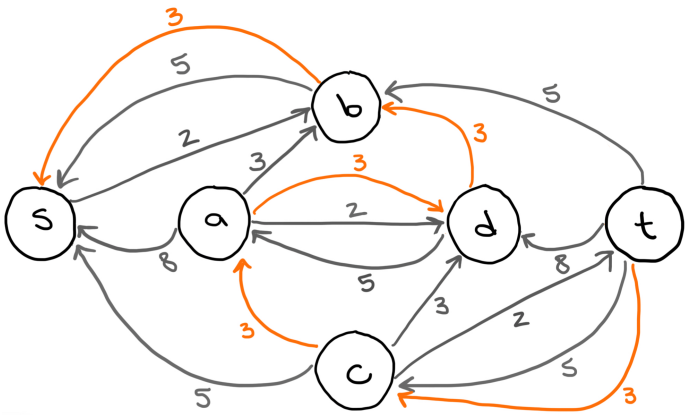


Augmenting path chosen: $s - c - t$; bottleneck capacity: 5; $f_3 : 13 \to 18$; residual graph:

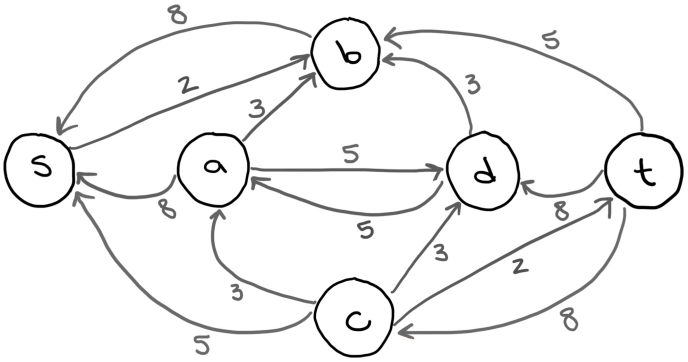


At first, it seems like we might have pushed as much flow as we can through the network...until we notice that we can actually send an additional 3 units through the path $s - b - d - a - c - t$.

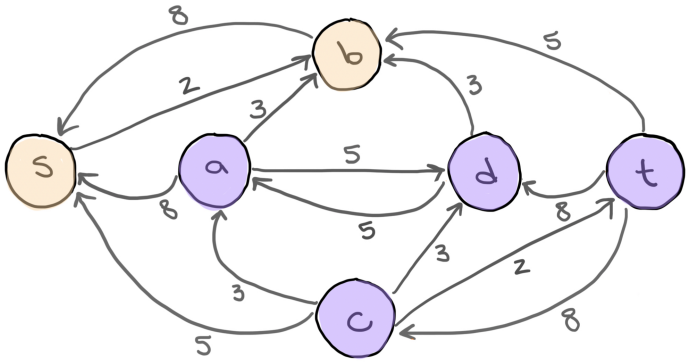
Augmenting path chosen: $s - b - d - a - c - t$; bottleneck capacity: 3; $f_4 : 18 \to 21$; residual graph:



At this point, we can send no additional flow through our network, giving us a max flow $f_{max} = 21$ and the following final residual graph:



We showed in lecture that the minimum cut of a graph can be computed by running Ford-Fulkerson and looking at the nodes reachable from s in the final residual graph. On the residual graph above, we see that the only other node reachable from s is b. Thus, $S = \{s, b\}$, $T = \{a, c, d, t\}$ is the minimum cut of the graph above. As a sanity check, we see that value of the cut S, T is 21 – which is also the value of the maximum flow we computed through our graph. Thus, our results are consistent with the max-flow min-cut theorem.



Problem 2

The final year project coordinator at a major university in Scotland would like to assign final year projects to the students. There is a set S of students, and a set P of projects which have been proposed by the supervisors, and each project j has a certain capacity q_j, meaning that it can be assigned to at most q_j students. To make the task of eliciting preferences from the students easier, the project coordinator has only

asked the students to rank each project as either "acceptable" or "unacceptable". In the end, each student should be assigned to at most one project, and a student cannot be assigned to a project which is unacceptable to them. For each student that is left unassigned, the project coordinator will lose one night of sleep.

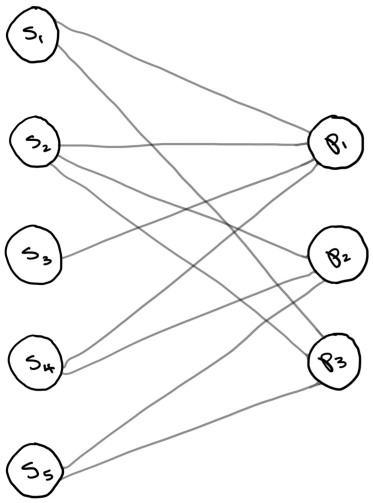
Design a polynomial-time algorithm for minimising the nights of sleep that the project coordinator will need to lose. To do that, reduce the problem described above to the problem of finding a maximal flow in a flow network. Provide an argument for the correctness of your reduction.

Solution

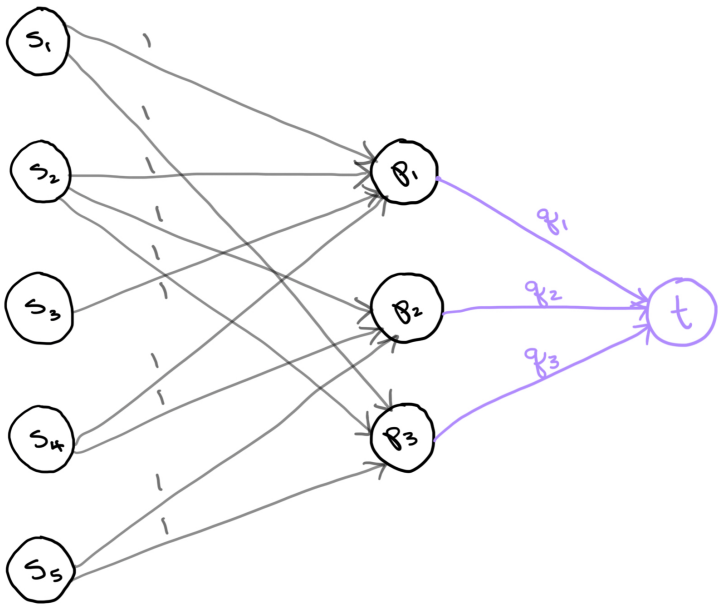
Let's break this down step-by-step. We know that we want to model the scenario above as a flow network, and this flow network must somehow capture all of the following pieces of information:

- A set S of students and a set P of projects
- Binary preferences of students for each project
- Capacities q_j for each project $j \in P$
- Assignments of at most 1 project to each student.

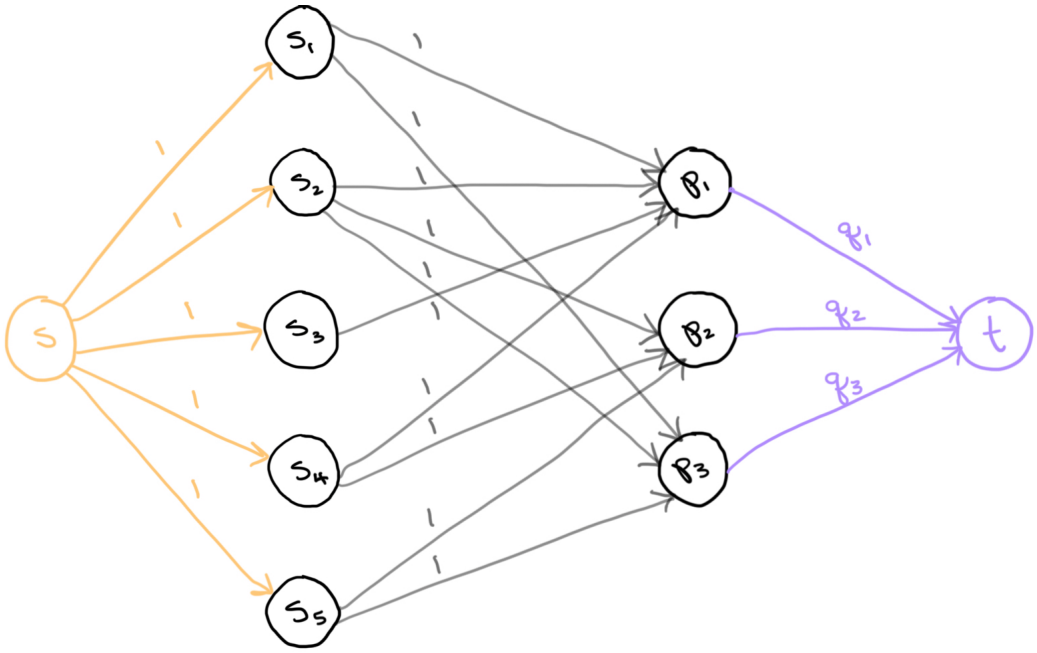
The first two elements naturally lend themselves to a bipartite graph, with an edge (i, j) from student i to project j if and only if the student finds that project acceptable. For instance:



Next, we need to somehow capture the project capacities. One idea would be to place the capacities q_j on each incoming edge to nodes p_j. But we quickly realise that this wouldn't work: For instance, node p_1 has three incoming edges. Giving each edge a capacity q_1 would allow $3q_1$ students to be mapped to project p_1 in a flow network. So our next idea is to add an outgoing edge from each project p_j to some additional node, say, t. We let each edge (p_j, t) has capacity q_j. Since in a flow network, a node's incoming flow must equal its outgoing flow, this forces at most q_j incoming flow to each node p_j. If we assign a capacity of 1 to each student-to-project edge, this means that at most q_j students can be assigned to project p_j in the corresponding flow network, which is exactly what we want.



Finally, we must consider our last requirement: that each student must be assigned to at most one project. Relying once more on the conservation of flow between incoming and outgoing edges of a node, we add an incoming edge of capacity 1 from some source node (say, s) and each student node. In this way, the incoming flow of at most 1 limits the outgoing flow of each student node to be at most 1, as desired.



To double check that our flow network provides an accurate model of the scenario we were given, we confirm that:

- Any flow in our network gives a valid student-to-project assignment.
- Any valid student-to-project assignment can be mapped to a flow in our flow network.

The first claim holds by the logic we used when constricting our flow network: Since there are only edges between students and acceptable projects, a student will only ever be assigned to an acceptable project. The

incoming edges of capacity 1 from s to each student ensures, through the conservation of flow through a node, that each student will only be mapped to at most project. The capacity constraints on the edges from each project to t similarly ensure that each project p_j will only have at most q_j students assigned to it.

Conversely, any valid assignment of students and projects can be represented in our flow network. The total amount of flow through the network equals the number of students assigned to projects, and there is a flow of 1 on an edge (s, s_i) if and only if student i is assigned to some project, a flow of 1 on edge (s_i, p_j) if and only if student s_i is assigned to project p_j, and a flow of $q'_j \leq q_j$ on edge (p_j, t) if and only if q'_j students are assigned to project p_j.

Now that we've modelled a flow network that captures all the relevent information from the problem statement, we can try to use this model to solve the original question.

- How can we find an allocation of students to projects which minimises the number of projectless students?

Since we've modelled our scenario as a flow network, our first guess/hope is that such an assignment corresponds to a maximum flow in our network. To show that a maximum flow minimizes the number of projectless students, we simply recall that a max flow is one in which the total amount of flow out of the source s is maximal. But the total amount of flow s corresponds to the number of students who are assigned to projects. Thus, the max flow of our network gives us a maximal assignment of students to projects.

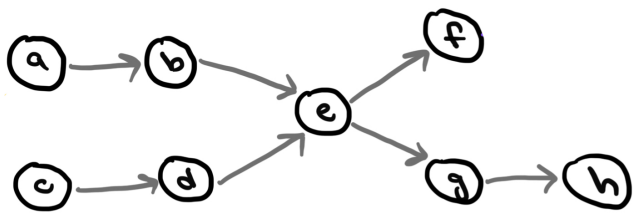
Problem 3

The timetabling team at a major university in Scotland would like to assign the current cohort of students to courses for the duration of their studies. The courses exhibit a pre-requisite relation, i.e., there are courses on which students can be assigned only if they have been assigned to some other courses that are prerequisites. One way to think of this is in terms of a graph, in which the nodes corresponds to the courses and a directed edge (i, j) denotes that course i is a prerequisite for course j. Luckily, the degree programme is well-designed, and thus the prerequisite relation graph is acyclic. Each course i has a capacity c_i on the number of students that can be assigned to it. To simplify the assignment process, the timetabling team has decided to do the assignment without allowing students to select courses. A student will not be assigned to any courses if it is not feasible for them to get a degree.

Design a polynomial-time algorithm for maximising the number of students that can eventually get a degree. To do that, reduce the problem described above to the problem of finding a maximal flow in a flow network. Provide an argument for the correctness of your reduction.

Solution

Starting off, we note that we're already given a directed, acyclic graph G which captures the prerequisite relations described in the problem statement. So let's try to construct our flow network using G as our starting point. For instance, we might have a graph G as follows:



There are multiple ways we might interpret the problem statement above – for instance, consider node e. Do its two incoming edges represent separate prerequisites b and d which must both be satisfied before taking

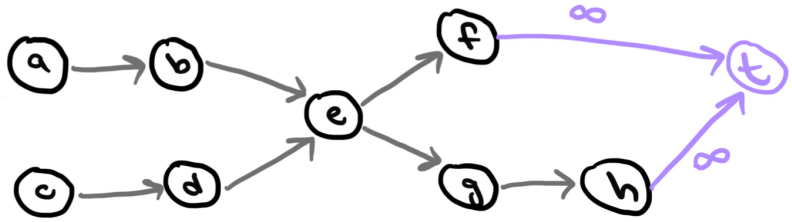
taking course e? Or need students take only one or the other? For simplicity, we assume that the former is true; i.e., students must take either b or d before taking course e. This assumption simplifies our initial modelling setup, but we could also restructure our graph to accommodate the other assumption if we wanted to.

As in problem 2, our next step is to list all the pieces of information from the problem statement we want our flow network model to capture.

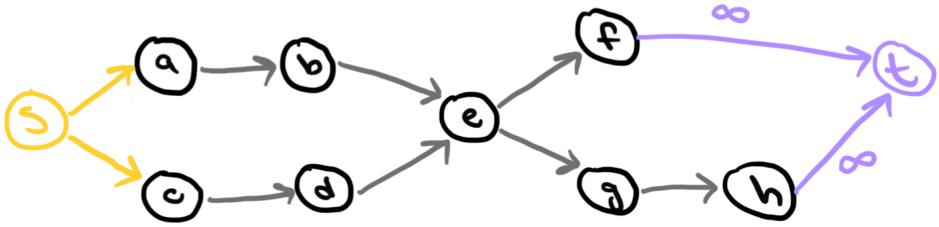
- There is prerequisite-relation graph.
- A degree program ends whenever a student completes a course which is not a prerequisite for some other course.
- Each student either is assigned to degree-completing path or to no path at all.
- Course capacities c_i for each course/node are shown in the graph.

How can we augment and/or alter our original graph G to include this information? We have the first item already. Let's consider the middle two points in our list above. The idea is that one unit of flow is one student, hence we will be "routing" students through the courses of their degree. The students will be initially assigned to the sources of the graph (the courses without pre-requisites) and then through the edges of the graph to compatible courses. Once they reach a final course, they can move directly to graduation, which is when the flow reaches the sink t.

We need to find some way to indicate whether a sequence of courses ends in a degree. For this, we can simply add a terminal node t, and an edge to t from each course which is not a prerequisite from some other course. What should the capacities of these edges be? Well, our problem statement doesn't (directly) cap the number of students who can graduate from a given sequence of courses, so why not give these edges infinite capacity? E.g.,

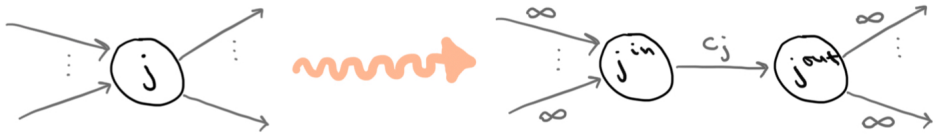


Now we need to capture the fact that students will be moving through this network, through G into t. For this, let's add a source node s and an edge between s and each "starting course"; i.e., course with no prerequisites.



Now we turn to the question of course capacities. Our first instinct might be to let the capacity of each incoming edge to some node i be c_i. If there is only a single incoming edge, then this approach works perfectly well. However, suppose a node has multiple incoming edges, like node e in our example. Setting the capacities of edges (b, e) and (d, e) both equal to c_e no longer works, since it allows an incoming flow of up to $2c_e$ into node e.

Instead, we use the following trick. For each course-node of the graph j, we *split* the node into j^{in} and j^{out}. We reroute every incoming edge to j to be incoming to j^{in}, and any outgoing edge from j to be outgoing from j^{out}. Finally, we add a directed edge (j^{in}, j^{out}) with capacity c_j. Now, it is ensures that the total amount of flow that will be routed via node j will be at most c_j, regardless of where the flow came from. We set all other edge capaticies in the graph equal to, say, infinity.



It follows from the construction that the value of the maximum flow is equal to the maximum number of students that can obtain their degree. We can compute a maximum flow using the Edmonds-Karp algorithm in time $O(|V| \cdot |E|^2)$.