FIT2004 - Algorithms and Data Structures

Seminar 9 - Circulation with Demands and Applications of Network Flow

Rafael Dowsley

5 May 2025

Agenda

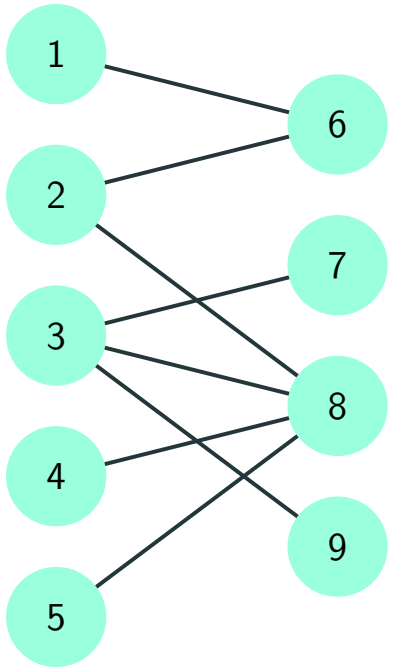
Divide-and-Conquer (W1-3)	Greedy Algorithms (W4-5)	Dynamic Programming (W6-7)	Network Flow (W8-9)	Data Structures (W10-11)

1. Maximum Bipartite Matching
2. Circulation with Demands
3. Applications of Network Flow

Maximum Bipartite Matching

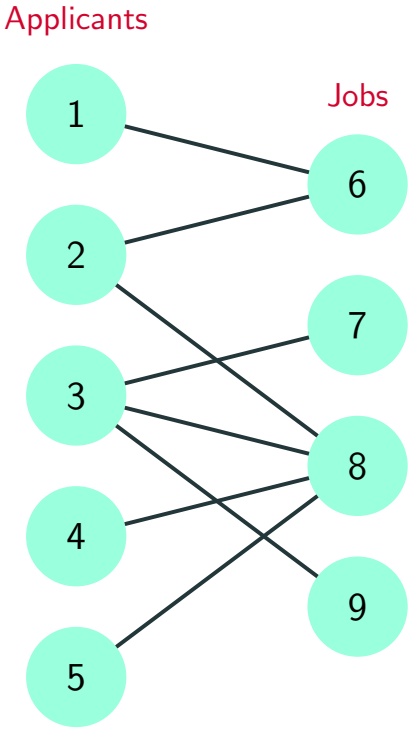
Bipartite graph

Bipartite graph: A graph where the vertices can be separated into two disjoint subsets L and R such that every edge connects a vertex $u \in L$ to a vertex $v \in R$.



Bipartite matching

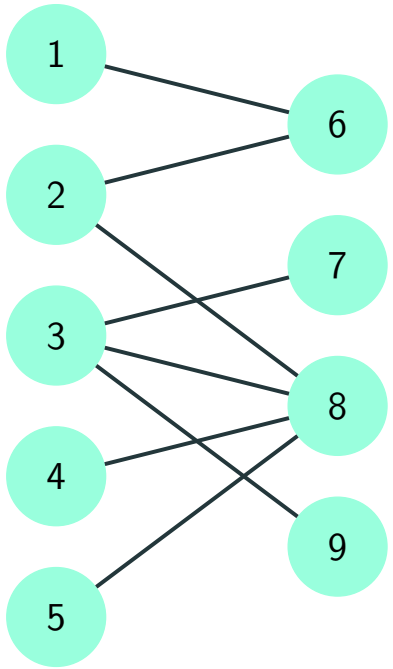
Suppose that the set L represents job applicants, and the set R are available jobs. There is an edge between the applicant u and the job v if person u is qualified for job v.



Goal: Match qualified applicants to jobs such that no applicant gets multiple jobs, and no job is taken by multiple people.

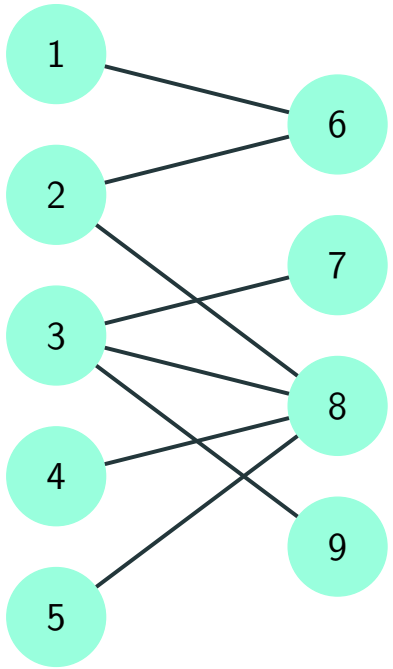
Bipartite matching

Bipartite matching: given a bipartite graph $G = (V, E)$ with $V = L \cup R$, a matching is a subset M of the edges E such that no vertex $v \in V$ is incident to multiple edges in M.



Maximum bipartite matching

Maximum bipartite matching: the maximum bipartite matching problem seeks a matching of the graph with the largest possible number of edges.

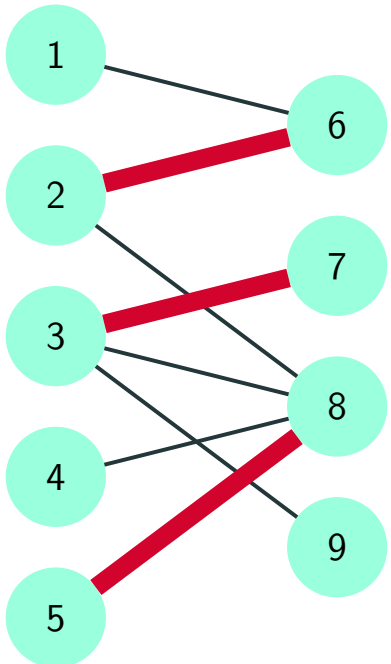


If $|L| = |R|$ and the matching has $|M| = |L|$, then it is called a perfect matching.

Quiz time!

Quiz answer

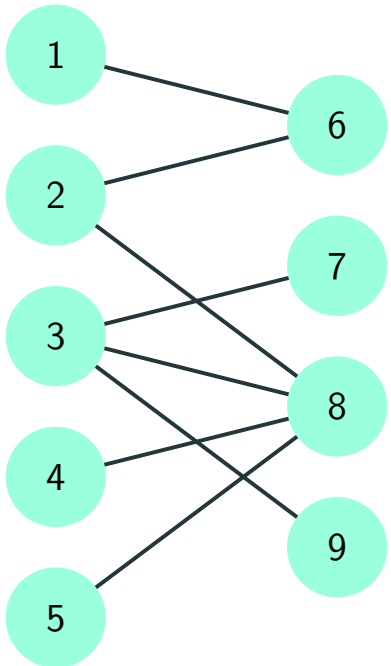
The maximum matchings for this bipartite graph have size 3. For example:



Note that vertices 7 and 9 are only connected to node 3.

How to solve it?

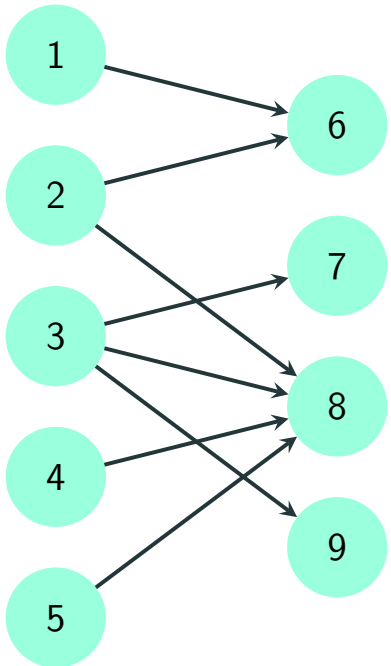
How to solve the maximum bipartite matching problem?



At first sight this problem may seem completely unrelated to the max-flow problem, but in fact...

How to solve it?

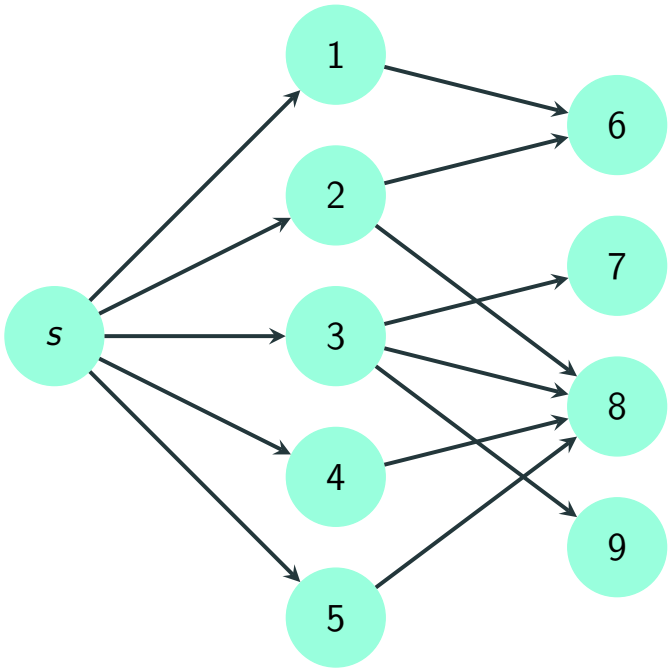
A corresponding flow network can be constructed, where the key idea is to use units of flow to represent matches:



Direct all edges to point from L to R.

How to solve it?

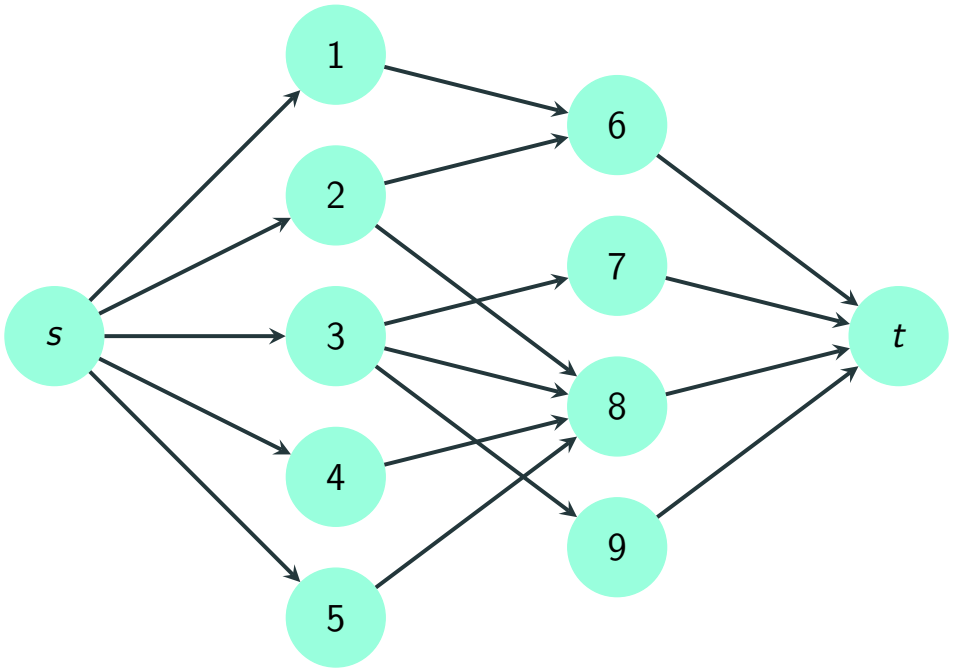
A corresponding flow network can be constructed, where the key idea is to use units of flow to represent matches:



Add a source s and connect it to all nodes in L.

How to solve it?

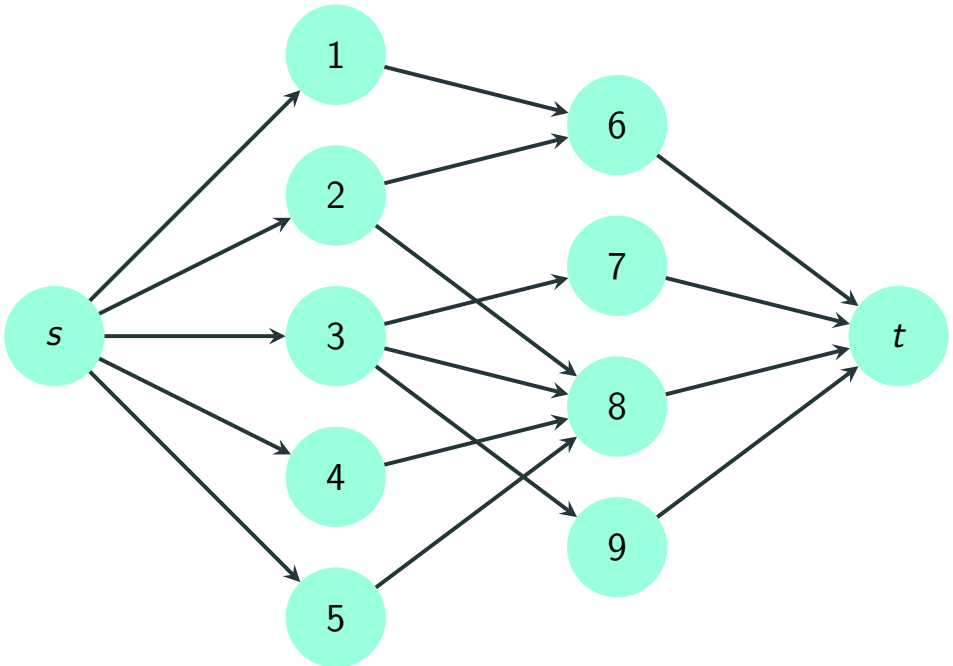
A corresponding flow network can be constructed, where the key idea is to use units of flow to represent matches:



Add a sink t and connect all nodes in R to it.

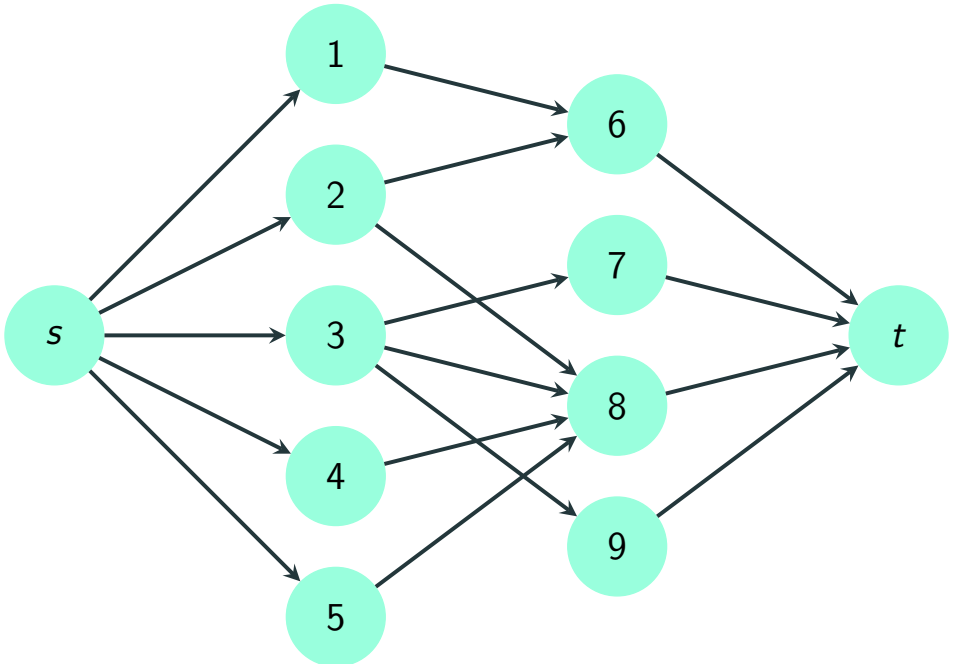
How to solve it?

A corresponding flow network can be constructed, where the key idea is to use units of flow to represent matches:



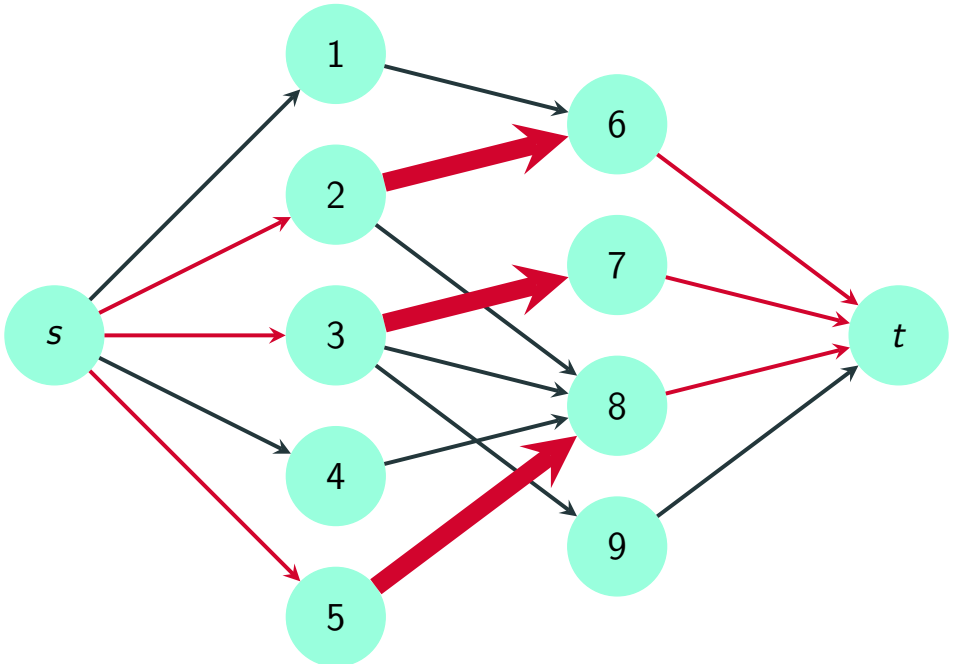
Give each edge a capacity of 1 (for simplicity will be omitted in the graph).

Why does this idea work?



- Since every edge has a capacity of one, at most one unit of flow can be sent from s through any vertex $u \in L$.
- For an integer-value flow, as the ones generated by Ford-Fulkerson, this ensures that for each node in L at most one outgoing edge is selected.
- Similarly, at most one unit of flow can leave any $v \in R$ into t, which ensures that for each node in R at most one incoming edge is selected.

The way to solve it



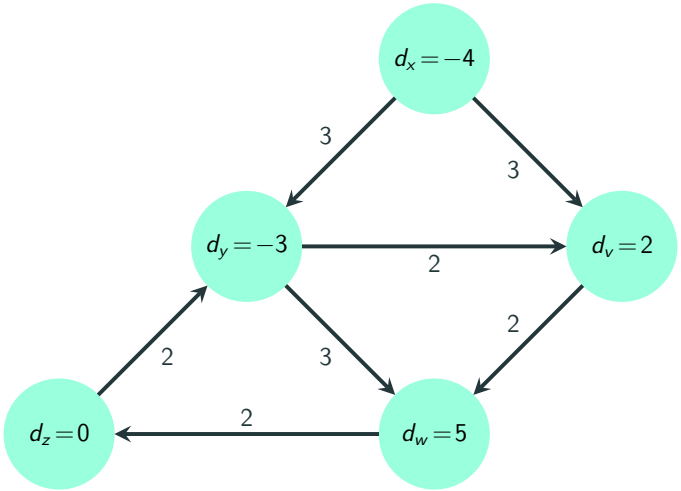
- Run Ford-Fulkerson to solve the max-flow problem.
- The maximum bipartite matching has size equal to the flow value.
- One solution to the maximum bipartite matching problem is given by the matches corresponding to edges from L to R that are saturated (i.e., flow of 1).

Circulation with Demands

Applications of Network Flow

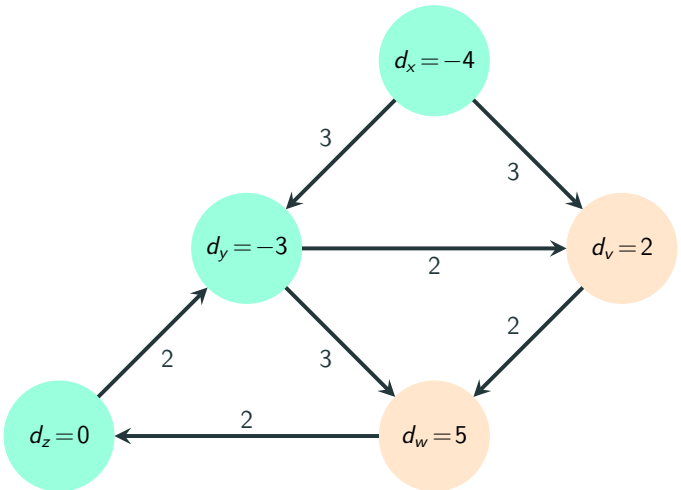
- The real power of network flow algorithms is that they allow us to efficiently solve non-trivial combinatorial problems that in principle have nothing to do with the flow of a material/data in a network.
- 1st example: solving the maximum bipartite matching problem.
- There are many further examples, but let's first introduce some generalisations of the max-flow problem that are often useful for solving this kind of problem.
- We will show that these generalisations can be reduced to the standard max-flow problem, and therefore the Ford-Fulkerson method can still be used.

Circulation with demands



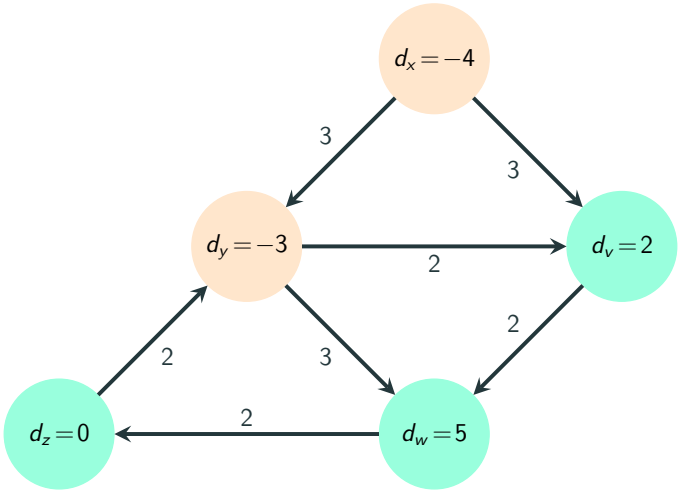
- In the problem of circulation with demands, there is no single source nor sink.
- **Every node can have both incoming and outgoing edges.**
- Each node u has an associated number d_u that denotes its condition.

Circulation with demands



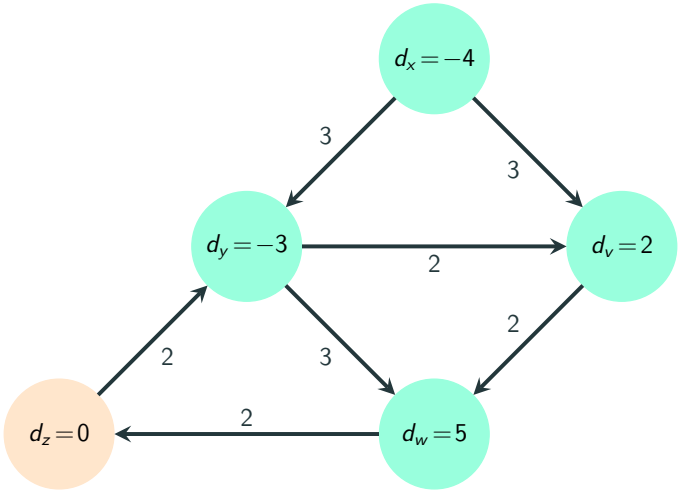
If $d_u > 0$, then node u has a demand of d_u units of flow, i.e., it wants to receive in its incoming edges d_u units of flow more than what it sends out in its outgoing edges.

Circulation with demands



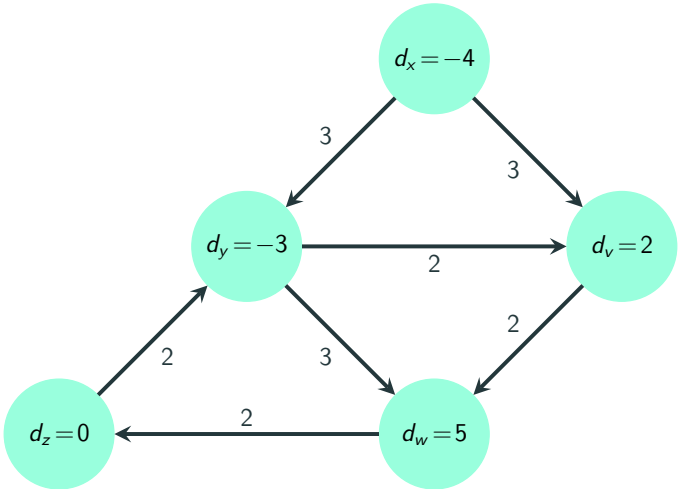
If $d_u < 0$, then node u has a supply of $-d_u$ units of flow, i.e., it wants to receive in its incoming edges d_u units of flow less than what it sends out in its outgoing edges.

Circulation with demands



If $d_u = 0$, then node u wants to keep the balance between the incoming and outgoing flow in its edges (similar to all non-sink non-source nodes in the standard max-flow problem).

Circulation with demands



- The circulation with demands problem is to determine the feasibility of satisfying these constraints together with the capacity constraints.

- A necessary condition for obtaining a feasible solution is that $\sum_{u \in V} d_u = 0$, but this is not a sufficient condition.

Properties of a circulation with demands

Formally, a circulation with demands $\{d_u\}$ should satisfy the following conditions:

Capacity constraint

For every edge e, its flow is bounded by its capacity: $0 \leq f(e) \leq c(e)$.

Demand constraint

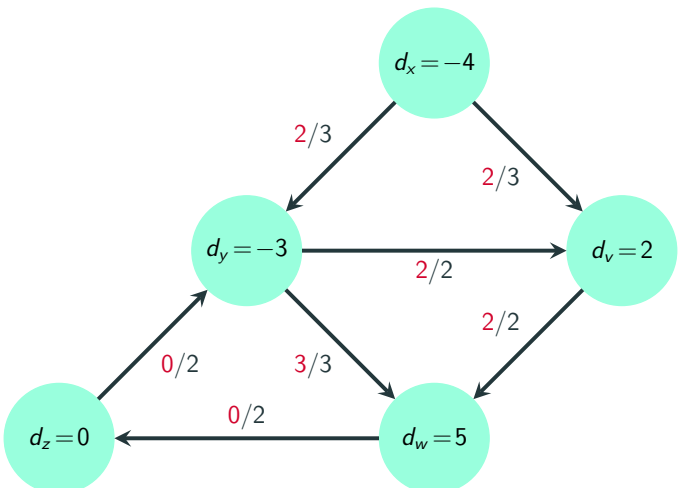
For every vertex $u \in V$, it holds that

$$\sum_{e_{in} \in E_{in}(u)} f(e_{in}) - \sum_{e_{out} \in E_{out}(u)} f(e_{out}) = d_u$$

We only consider integer capacities and demands.

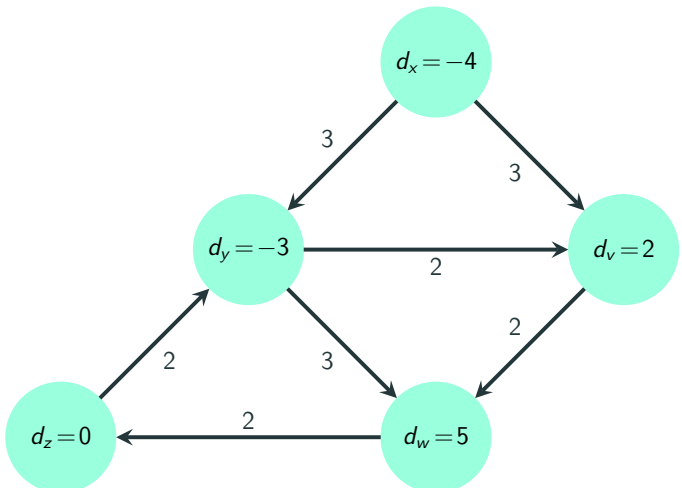
Is it feasible?

- **Circulation with demands:** a feasibility problem concerned with whether there exists a solution satisfying both constraints or not.
- But how can we get to this solution to our example?
- Idea: reduce the circulation with demands problem to a standard max-flow problem so that we can use Ford-Fulkerson to solve it.



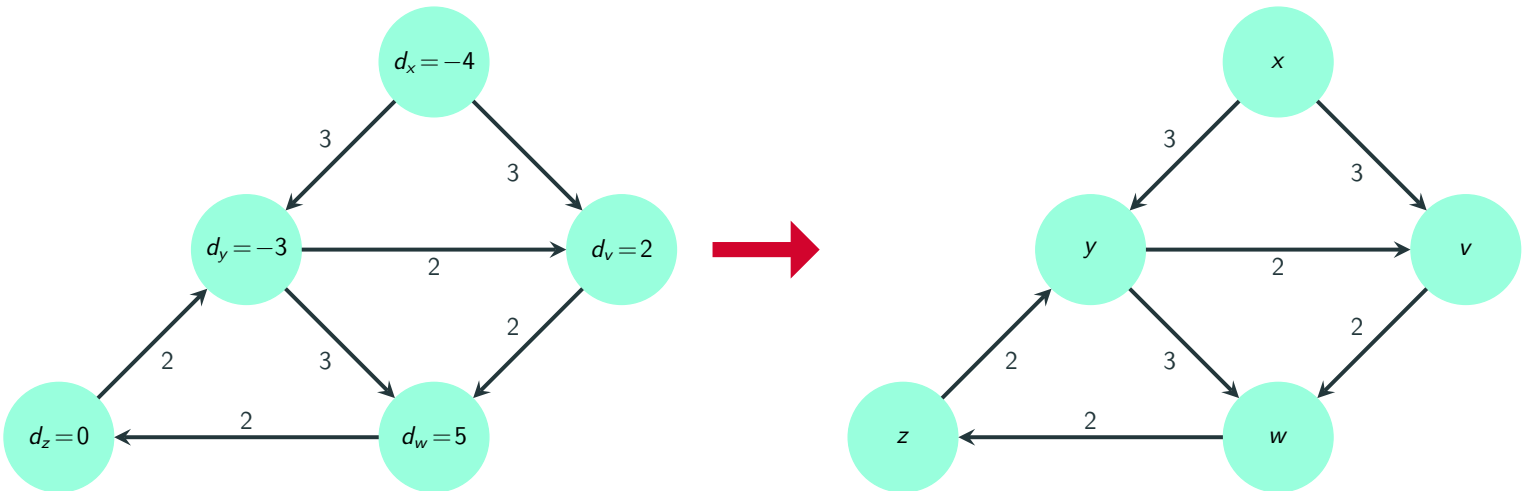
Solving it

The idea is to create a supergraph G' of G, solve a max-flow problem in G' getting a solution f', and then translate f' to a solution to the circulation with demands $\{d_u\}$ problem in G.



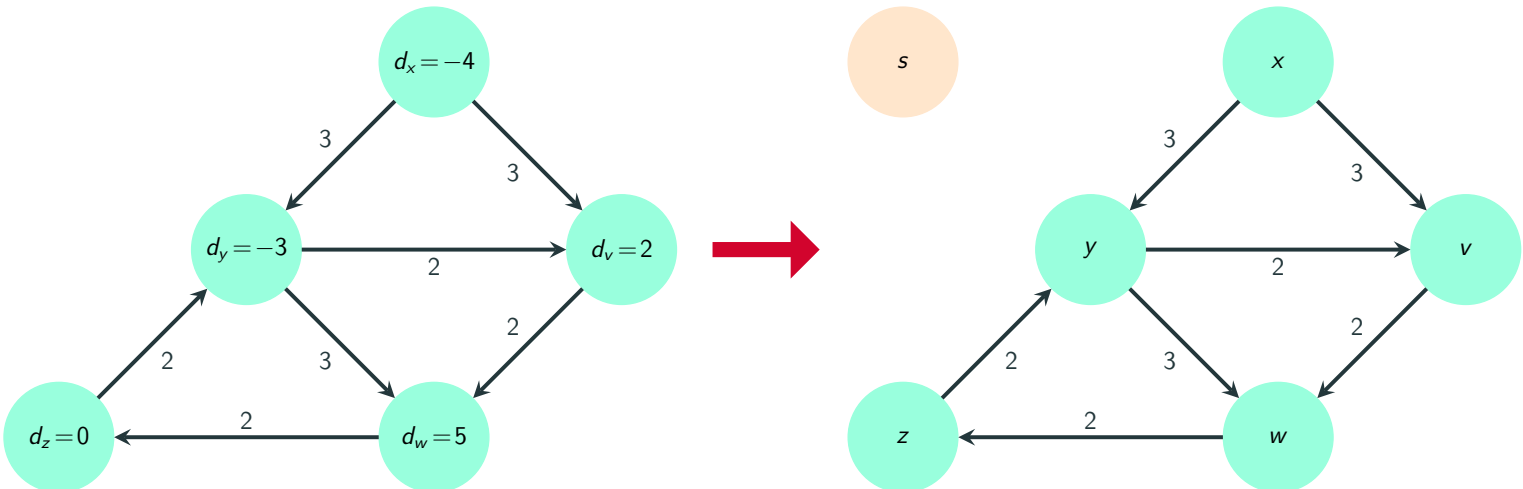
Solving it

Add each node and edge of G in G', but in G' the nodes do not have demands.



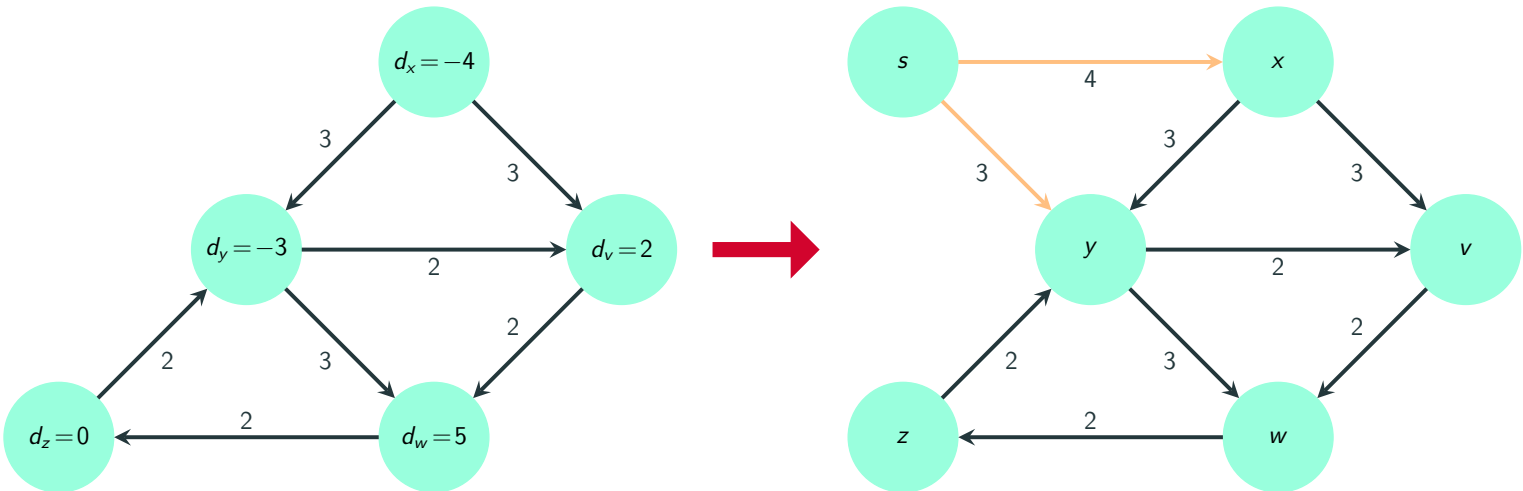
Solving it

Add a super-source node s.



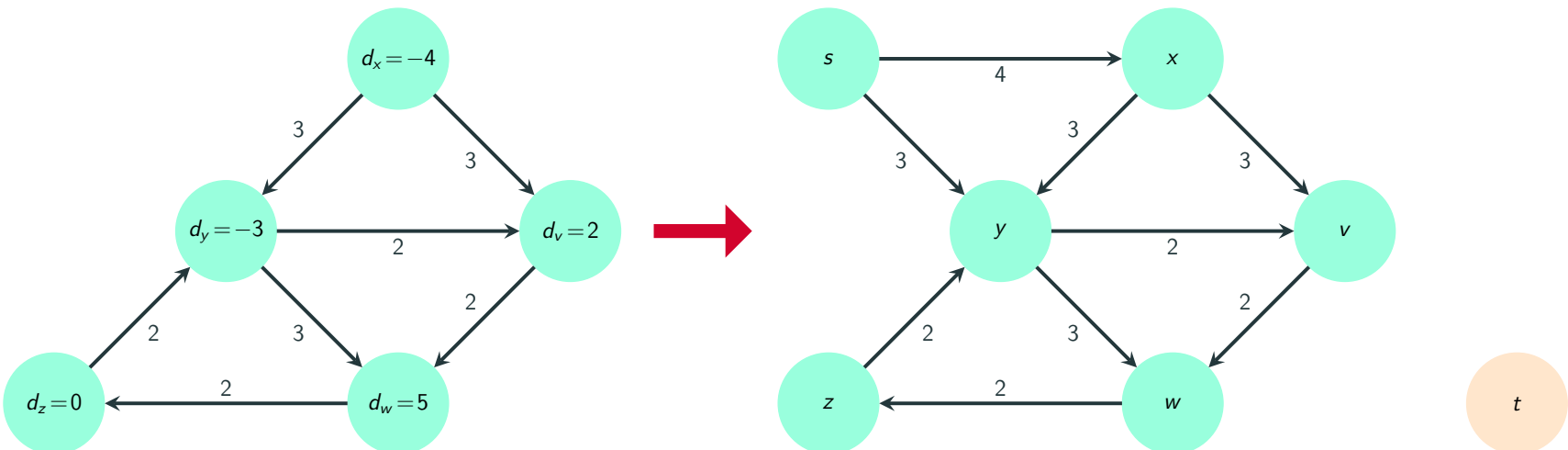
Solving it

For each node $u \in V$ such that $d_u < 0$, add an edge $s \to u$ to G' with capacity $-d_u$.



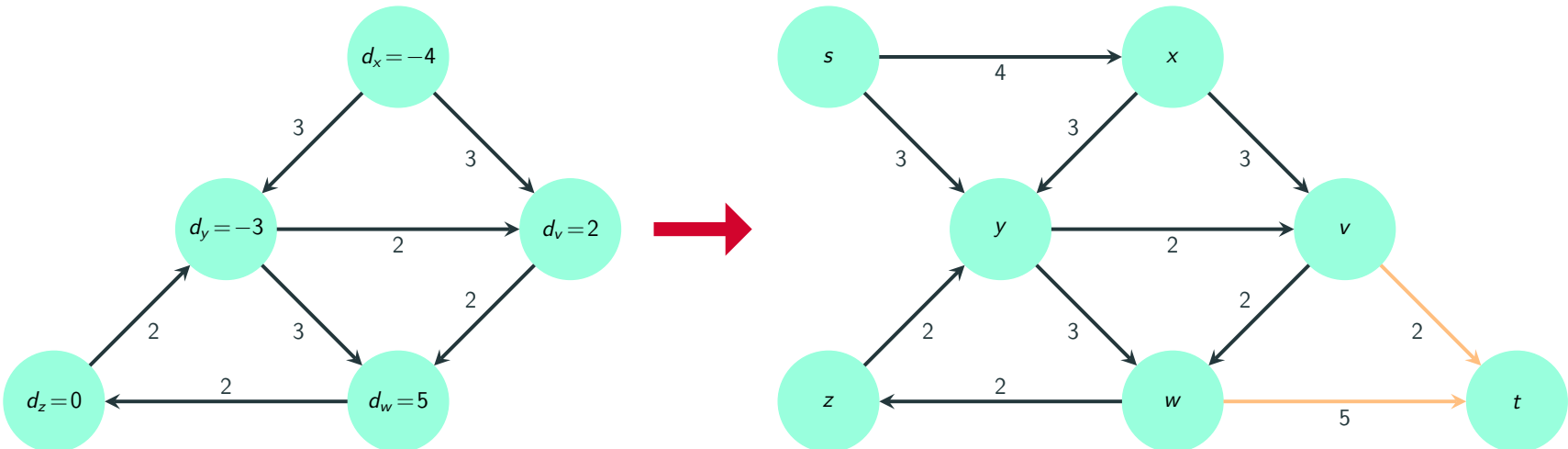
Solving it

Add a super-sink node t.



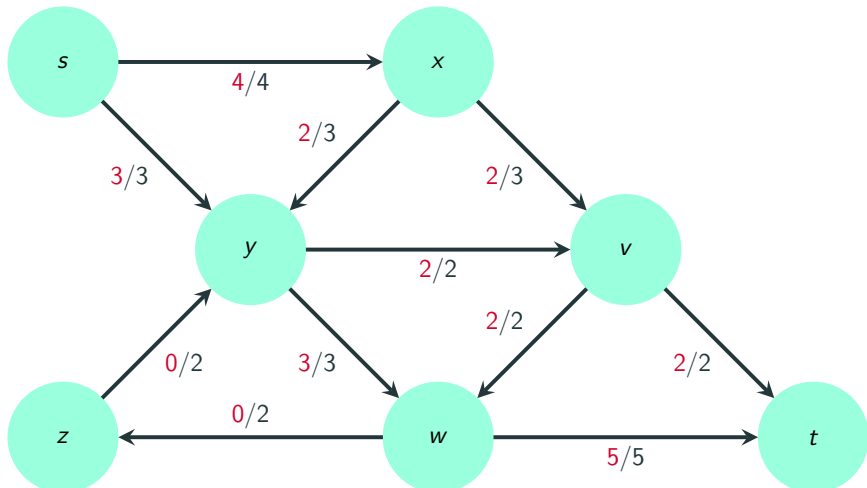
Solving it

For each node $u \in V$ such that $d_u > 0$, add an edge $u \to t$ to G' with capacity d_u.



Solving it

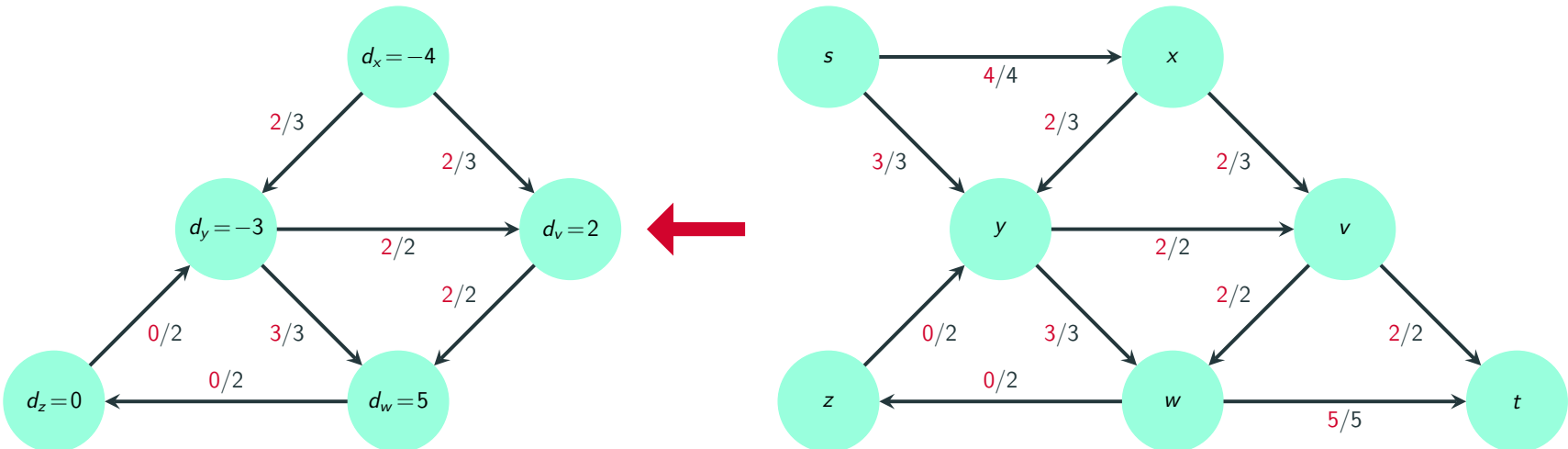
Solve the max-flow problem in G' using Ford-Fulkerson...



Solving it

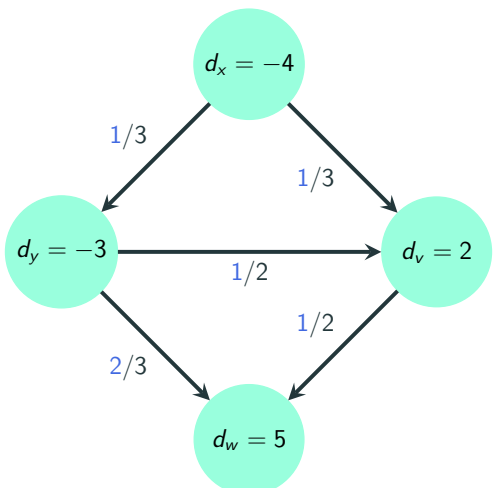
The circulation with demands in G has a feasible solution if, and only if, the solution of the max-flow problem in G' saturates all outgoing edges of s and all incoming edges of t.

If that is the case, a feasible solution can be found by just deleting the additional nodes and edges of G'.



Edges with lower bounds

- For many applications of network flow, it is also useful to consider edges with lower bounds.
- I.e., we will enforce that at least a specified amount of flow go through the edge.
- For each edge e, its lower bound $\ell(e)$ is denoted in blue in the following example.



Properties of a circulation with demands and lower bounds

Formally, a circulation with demands $\{d_u\}$ and lower bounds should satisfy the following conditions:

Capacity constraint

For every edge e, its flow is in between its lower bound and capacity: $\ell(e) \leq f(e) \leq c(e)$.

Demand constraint

For every vertex $u \in V$, it holds that

$$\sum_{e_{in} \in E_{in}(u)} f(e_{in}) - \sum_{e_{out} \in E_{out}(u)} f(e_{out}) = d_u$$

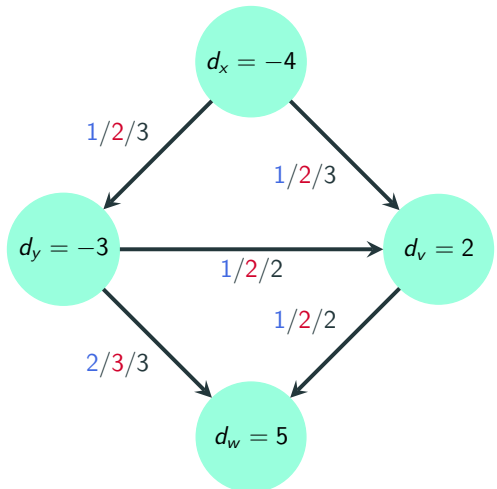
We only consider integer capacities, demands and lower bounds.

Circulation with demands and lower bounds

- **Circulation with demands and lower bounds:** a feasibility problem concerned with whether there exists a solution satisfying both constraints or not.

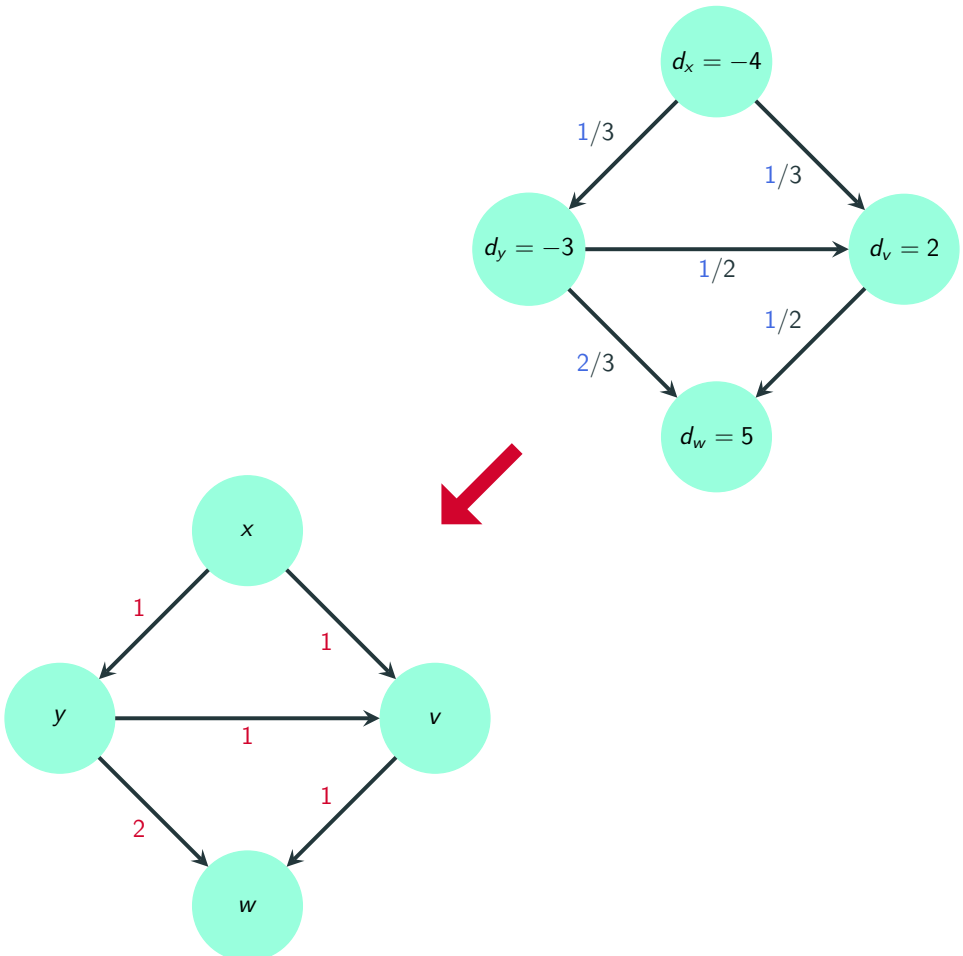
- How do we solve a circulation with demands and lower bounds problem?

- Idea: reduce it to a circulation with demands but no lower bounds. . .



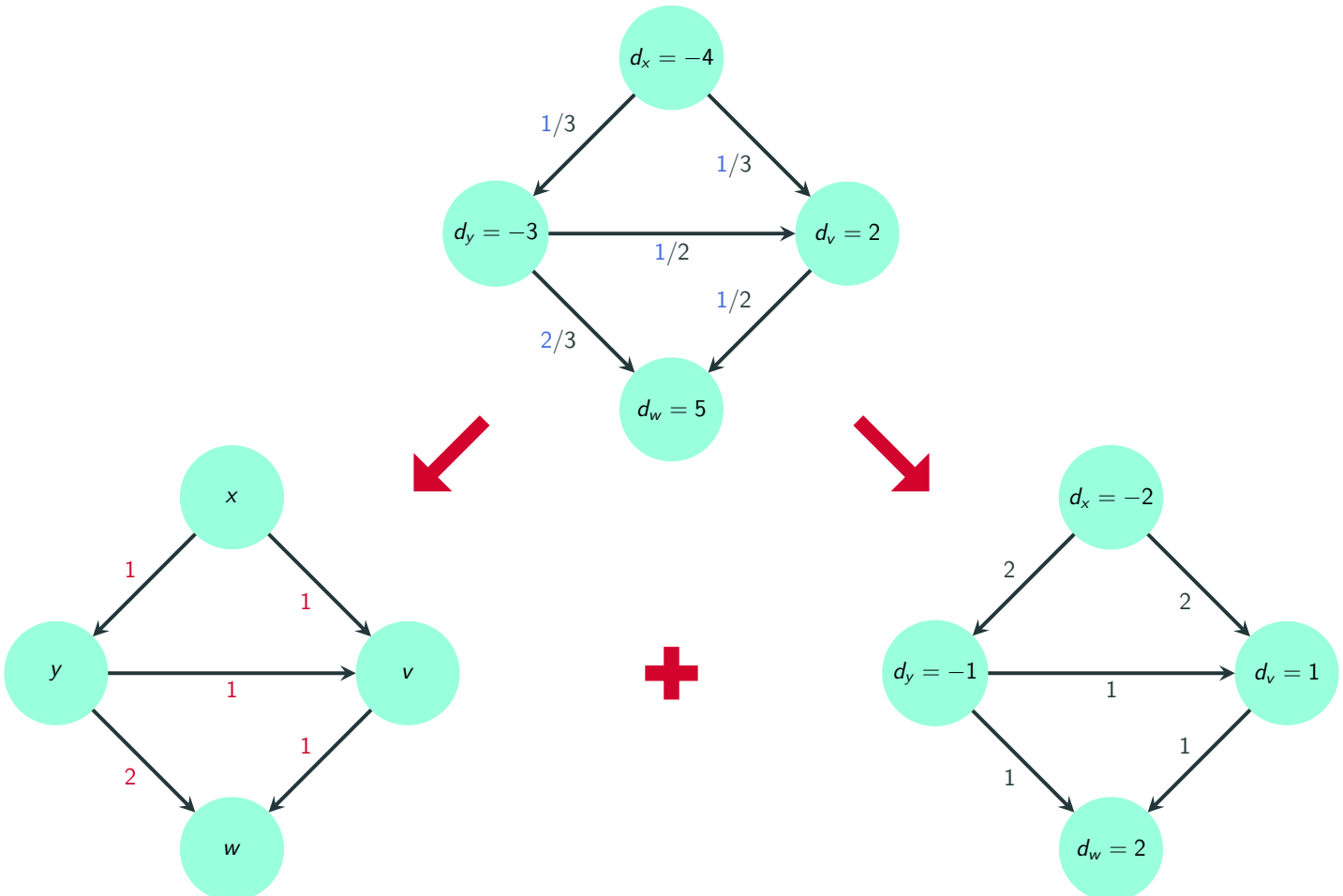
Reducing it

We will break the solution into a fixed flow that satisfies the lower bounds...



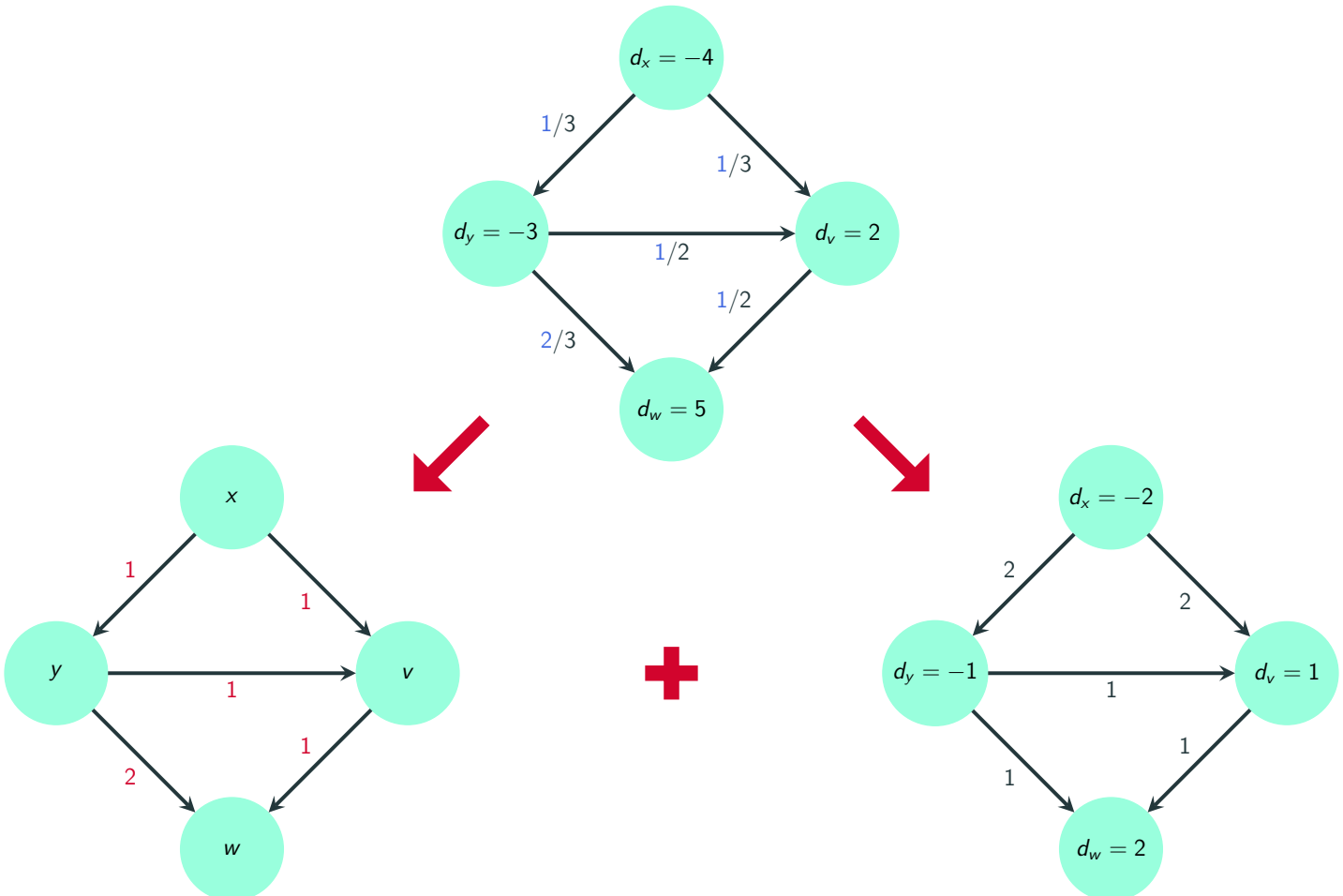
Reducing it

. . . plus a circulation with demands but without lower bounds on a graph with adjusted capacities and demands (to account for the fixed flow).



Reducing it

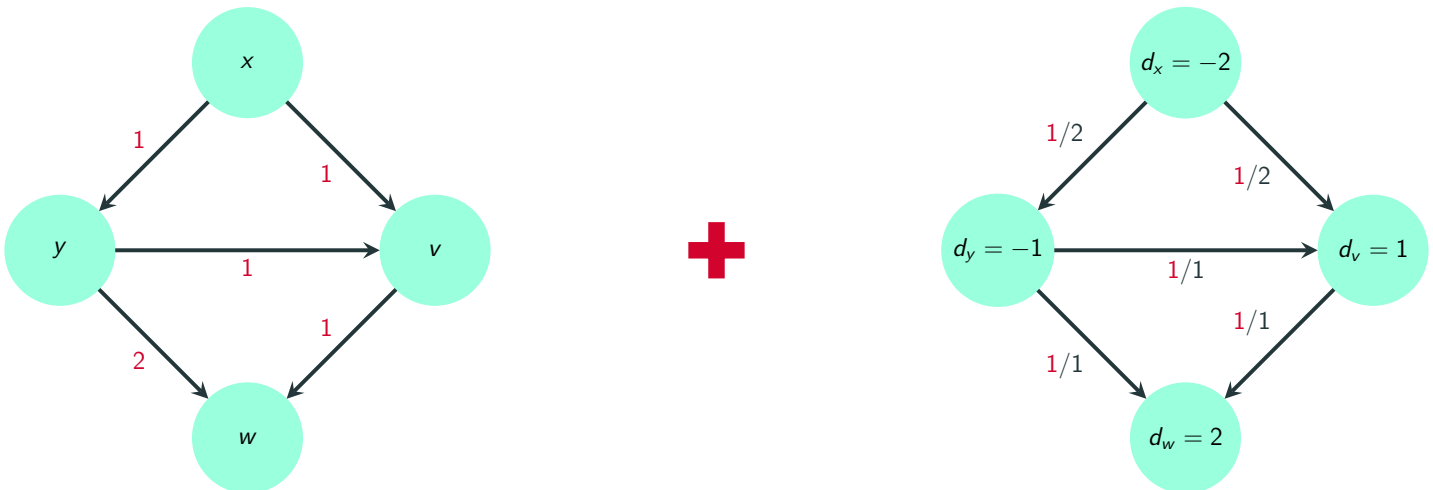
The original problem has a feasible solution if, and only if, the adjusted circulation problem (without lower bounds) has a feasible solution.



Reducing it

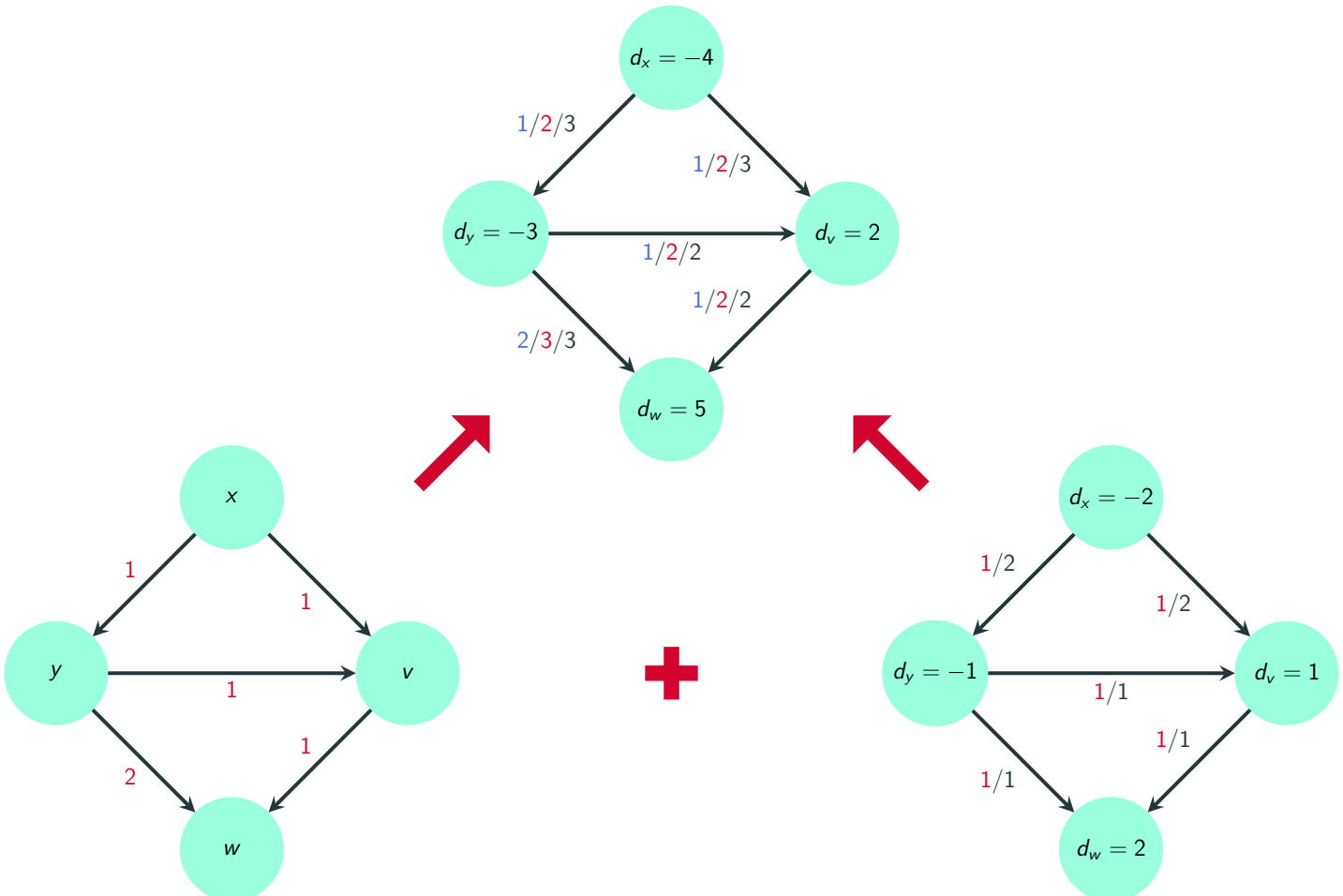
Solving the adjusted circulation problem using the method we learned before...

We get that there is a feasible solution, so the original problem also has a feasible solution that can be obtained by just summing the fixed flow to the solution of the adjusted problem.



Reducing it

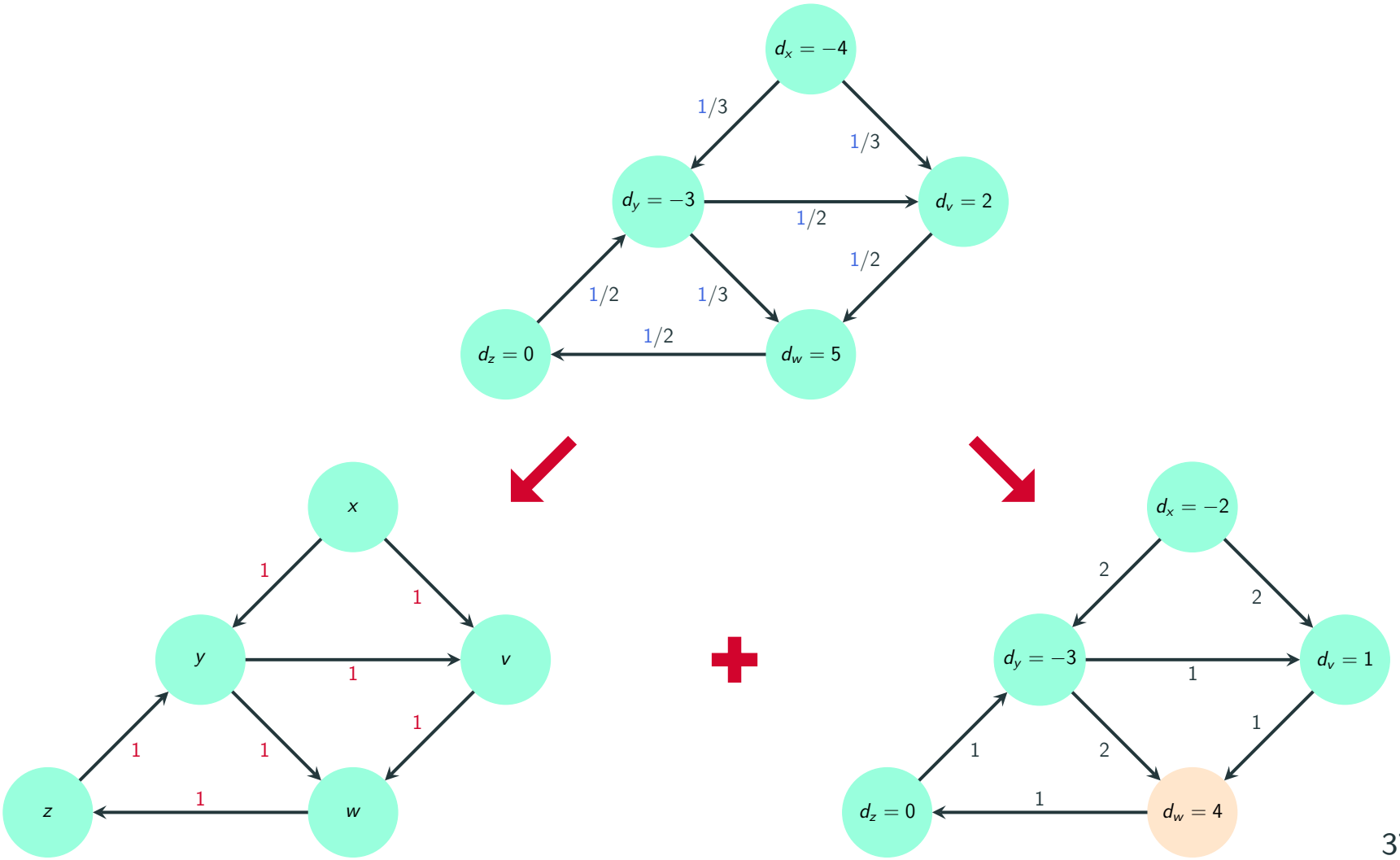
Getting a feasible solution to the original problem...



Quiz time!

Quiz answer

No feasible solution. **Demand on w cannot be satisfied.**



Applications of Network Flow

Integer-value solutions

- Remember that we only consider integer capacities, lower bounds and demands.

- **In this scenario, Ford-Fulkerson generates integer-value solutions, which is very useful in many applications.**

Survey design

- Your company sells some products P_j and wants to develop a customer survey to get feedback about them.
- You have a database with your customers C_i and the products they have bought.
- **Questionnaire size constraint:** to have reasonably-sized questionnaires, each client C_i should be asked about a number of products between C_i^- and C_i^+. Of course, a customer can only be asked about a product that s/he bought before.
- **Feedback size constraint:** to keep the survey informative, for each product P_j, between P_j^- and P_j^+ customers should be asked about it.

Survey design

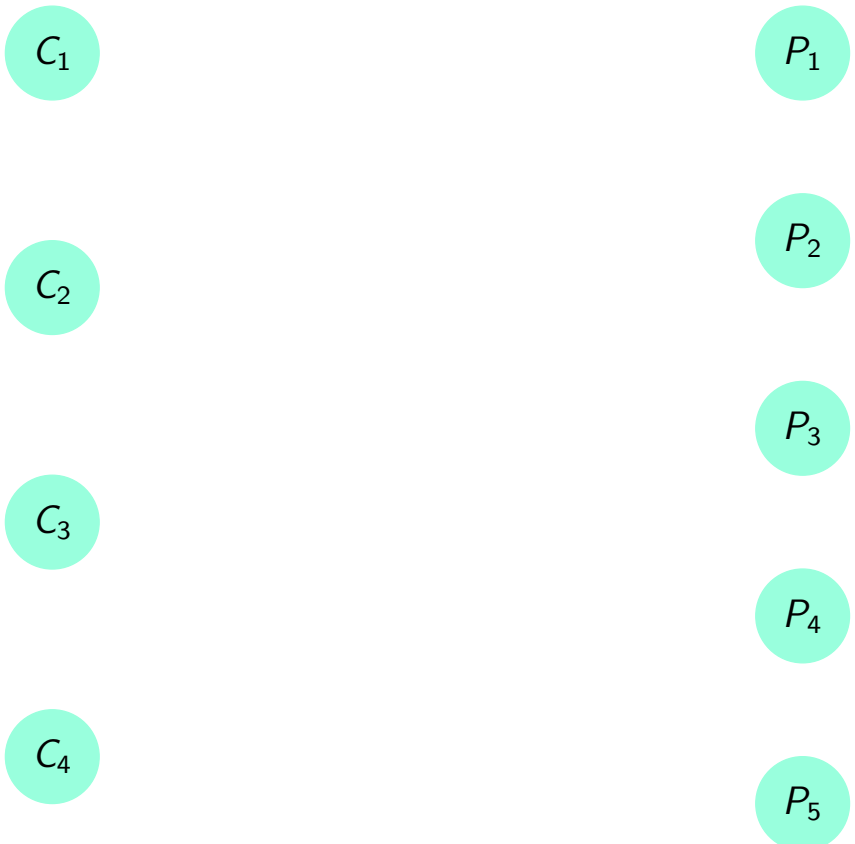
- Problem: Can your company design a survey meeting the constraints?

- We will express the survey design problem as a circulation with demands and lower bounds problem and check if there is a feasible solution.

- In fact, all nodes will have demand equal to 0, and therefore the demands will be omitted from the representation in the following slides.

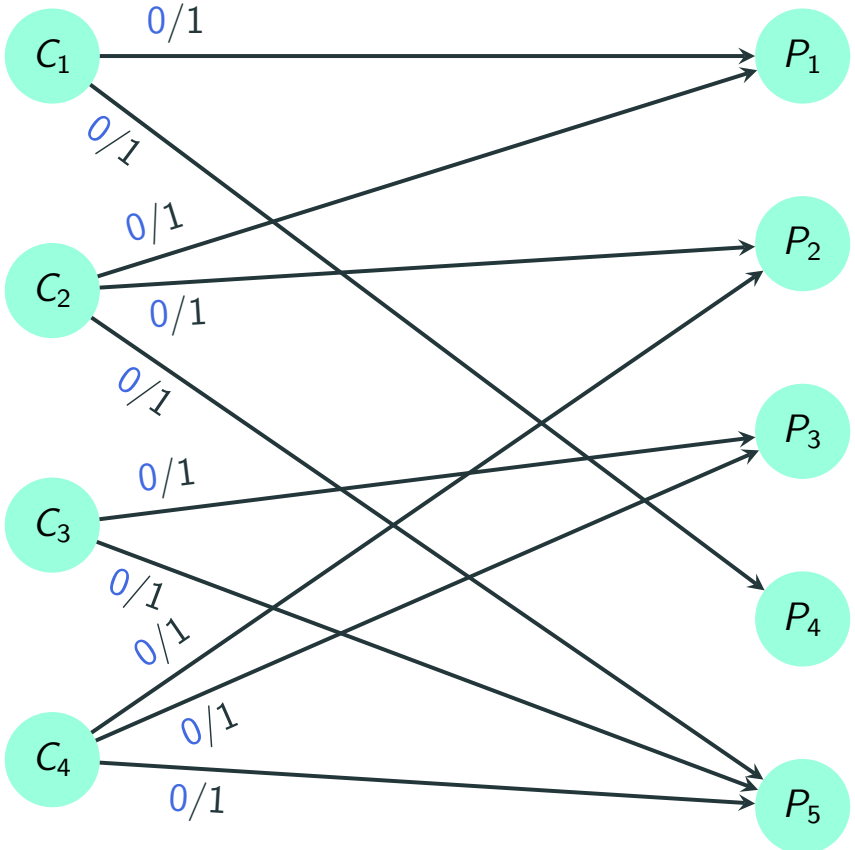
Survey design

Add one node to represent each customer and each product.



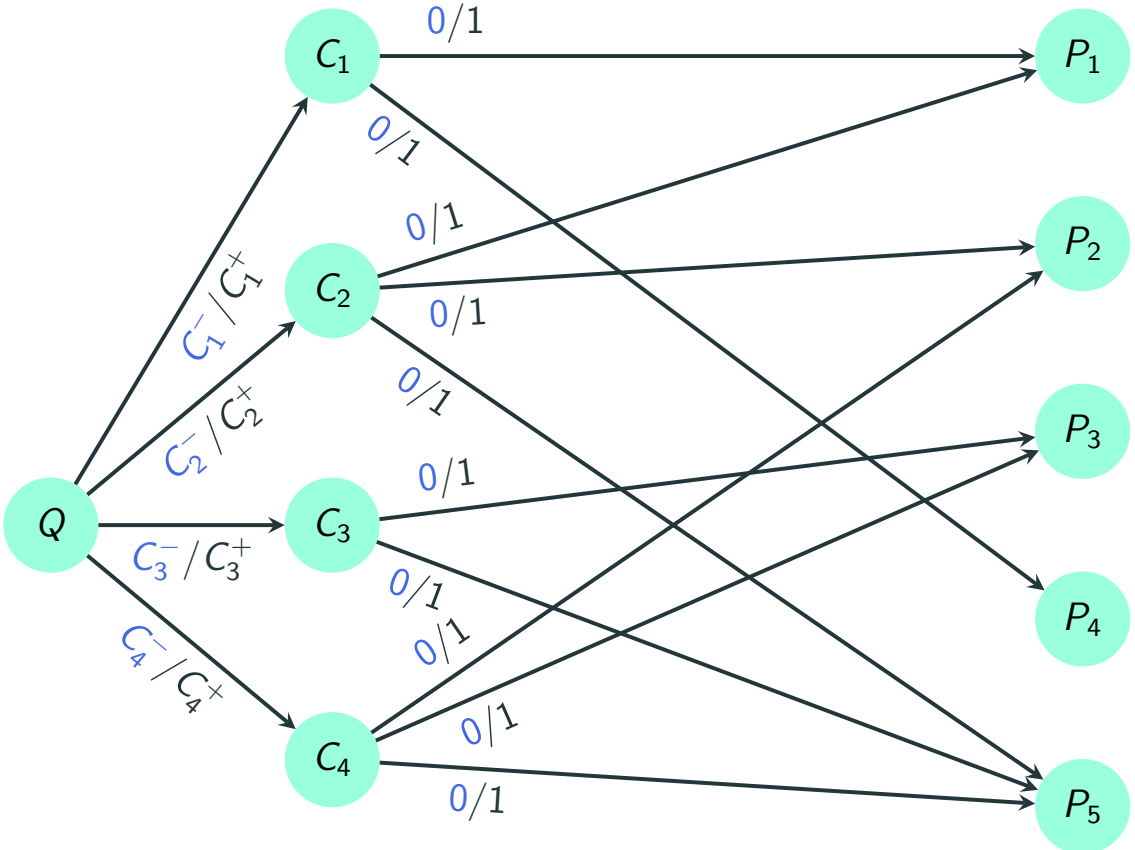
Survey design

For each product P_j that a customer C_i has bought, add an edge from C_i to P_j with capacity 1 and lower bound 0 (you can either ask the client's feedback about that product or not).



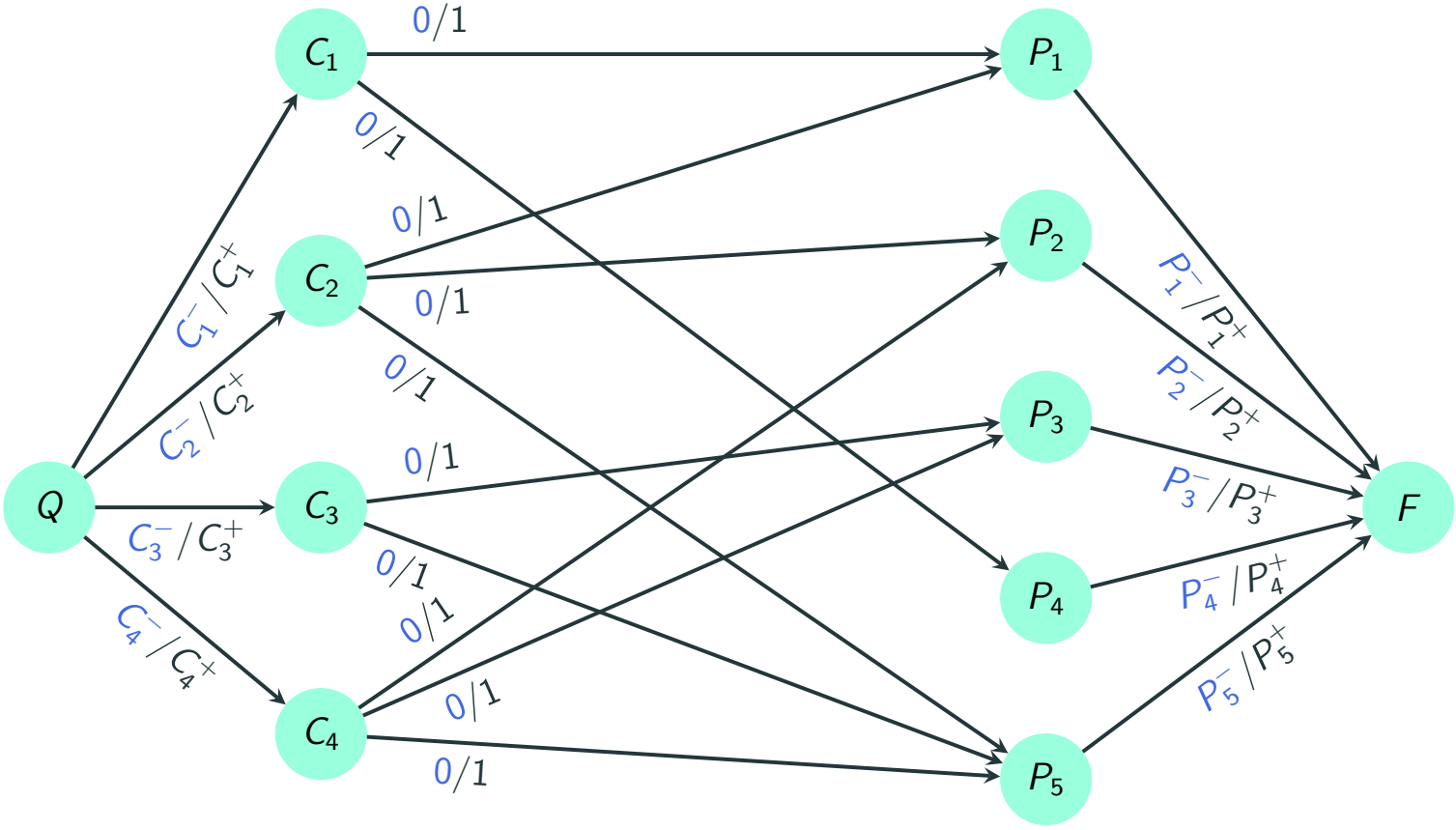
Survey design

Add a node Q representing the questions to be allocated. For each customer C_i, add an edge from Q to C_i with capacity C_i^+ and lower bound C_i^- (corresponding to the number of questions asked for that customer).



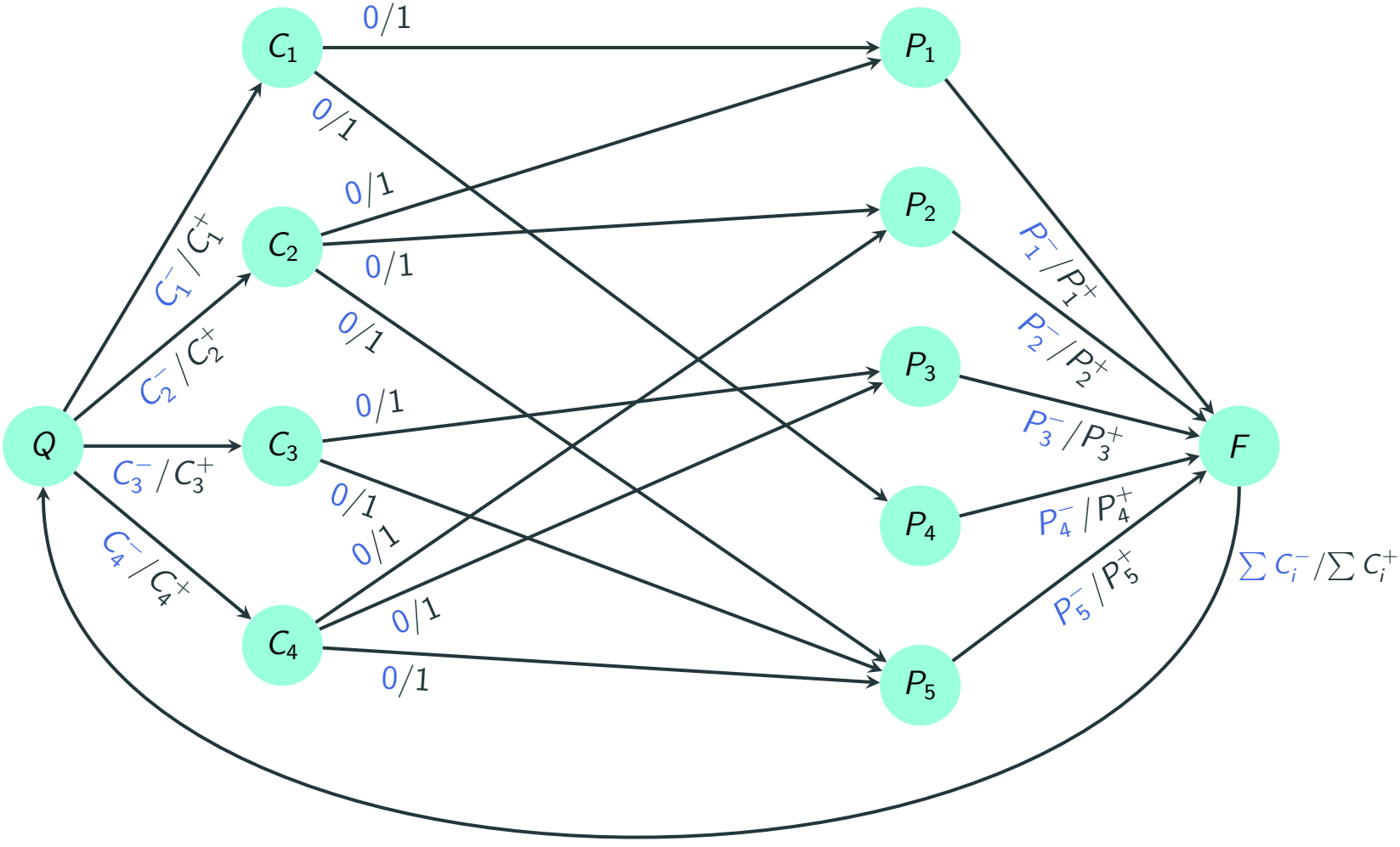
Survey design

Add a node F representing the feedback. For each product P_j, add an edge from P_j to F with capacity P_j^+ and lower bound P_j^- (corresponding to the number of customers asked about that product).



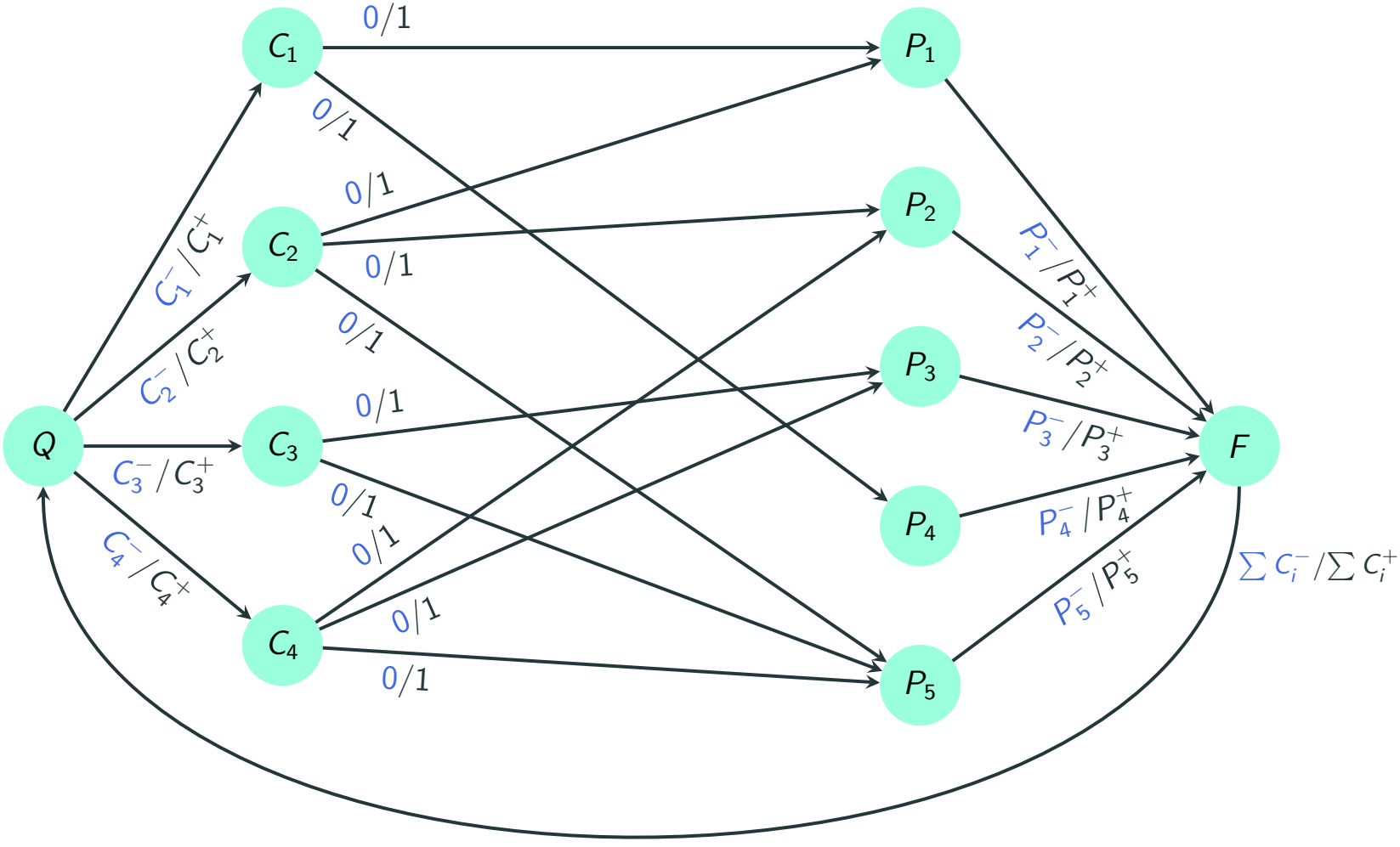
Survey design

Add an edge from F to Q with capacity $\sum C_i^+$ and lower bound $\sum C_i^-$ (corresponding to the overall number of questions asked).



Survey design

Solve this circulation with demands and lower bounds problem to check if there is a feasible solution to your survey design problem.



Airline scheduling

- Your company has k airplanes and is interested in flying a set of profitable flights during the day.
- You are given a list of j desirable flights (with their specified origin, destination, departure and arrival times).
- You also know which of those flights are reachable from each other (i.e., there is enough time for maintenance, boarding, fly the plane to another airport if needed to cover the next desirable flight etc).
- The planes can start and finish the day at any airport.
- Problem: Can your company cover the j desirable flights with the k planes?

Airline scheduling

- Flight 1: SYD 6am – MEL 7am
- Flight 2: CBR 8am – SYD 9am
- Flight 3: MEL 11am – BNE 1pm (reachable from flights 1 and 2)
- Flight 4: PER 11am – SYD 7pm (reachable from flight 1)

Is it possible to cover these 4 flights with only 2 planes? Yes:

- Airplane 1: Flight 1, fly from MEL to PER, Flight 4
- Airplane 2: Flight 2, fly from SYD to MEL, Flight 3

How can we solve this problem using network flow?

Quiz time!

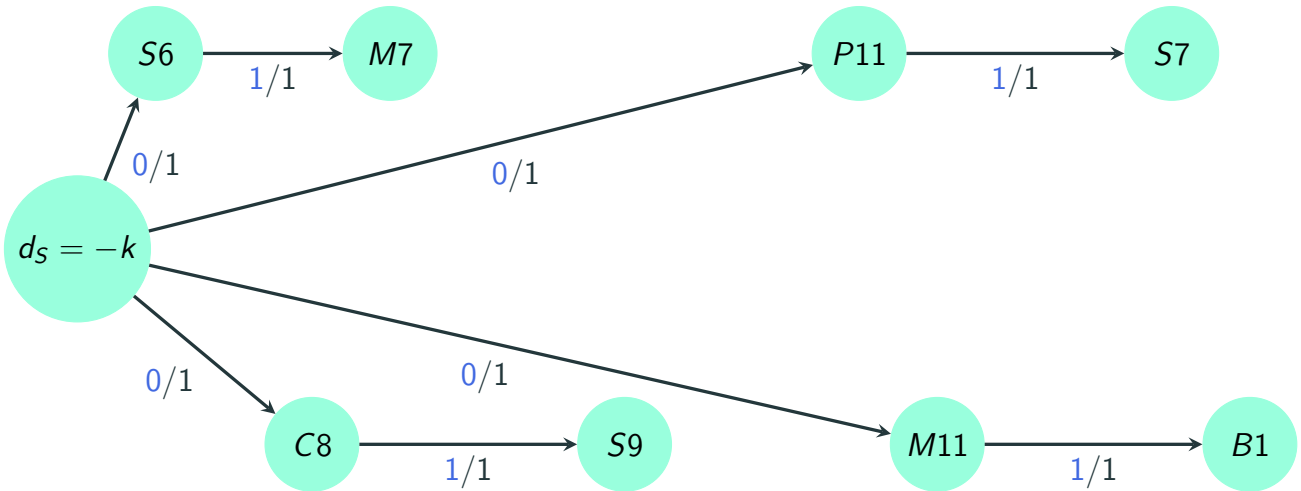
Airline scheduling

- Flight 1: SYD 6am – MEL 7am
- Flight 2: CBR 8am – SYD 9am
- Flight 3: MEL 11am – BNE 1pm (reachable from flights 1 and 2)
- Flight 4: PER 11am – SYD 7pm (reachable from flight 1)

For every desirable flight, add departure and arrival nodes, and an edge between them with capacity and lower bound 1 (i.e., it is necessary to fly the flight). All these nodes will have demand 0 (omitted for readability).

Airline scheduling

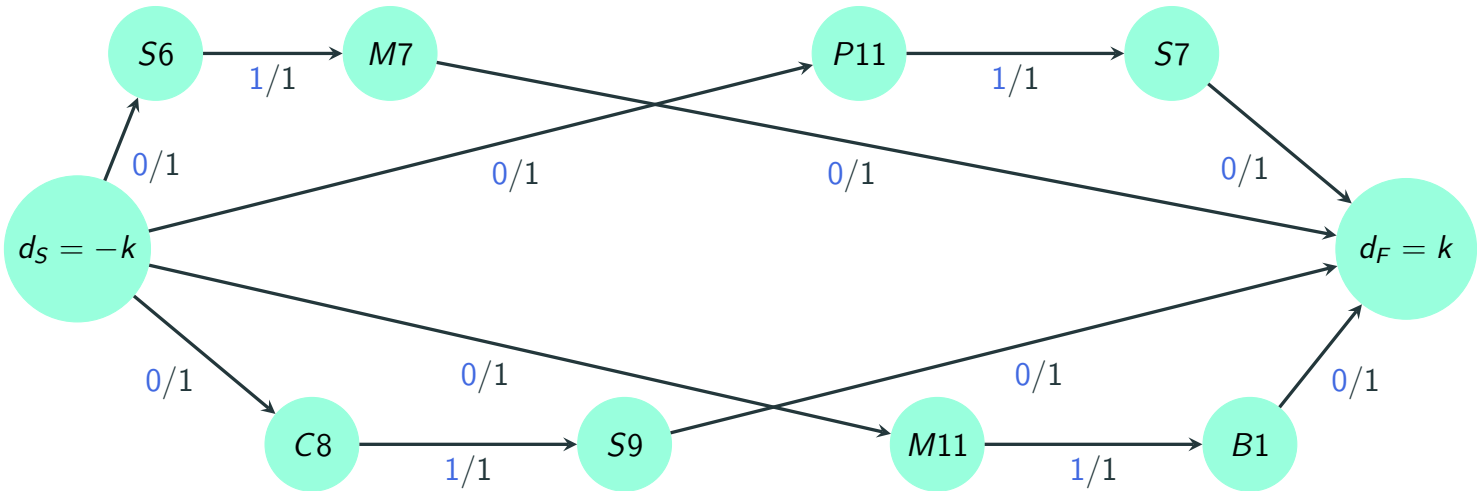
- Flight 1: SYD 6am – MEL 7am
- Flight 2: CBR 8am – SYD 9am
- Flight 3: MEL 11am – BNE 1pm (reachable from flights 1 and 2)
- Flight 4: PER 11am – SYD 7pm (reachable from flight 1)



Add a node S with demand $-k$ to represent the start of the day (that many planes are available). Add edges with lower bound 0 and capacity 1 from S to every departure node (planes can start with any flight).

Airline scheduling

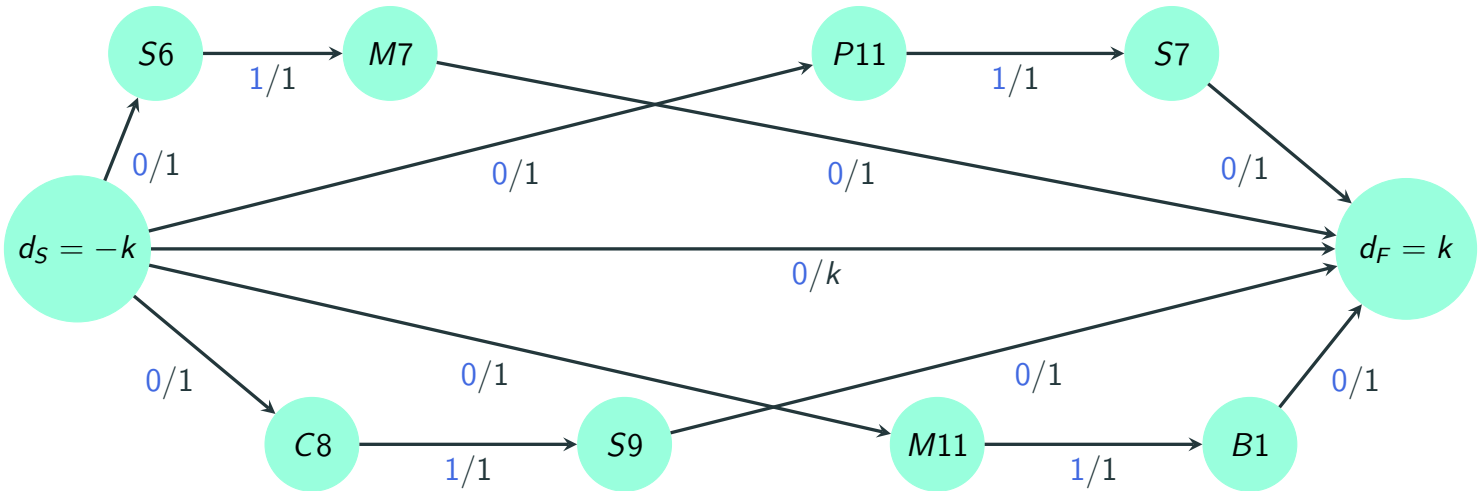
- Flight 1: SYD 6am – MEL 7am
- Flight 2: CBR 8am – SYD 9am
- Flight 3: MEL 11am – BNE 1pm (reachable from flights 1 and 2)
- Flight 4: PER 11am – SYD 7pm (reachable from flight 1)



Add a node F with demand k to represent the end of the day. Add edges with lower bound 0 and capacity 1 from every arrival node to F (planes can finish with any flight).

Airline scheduling

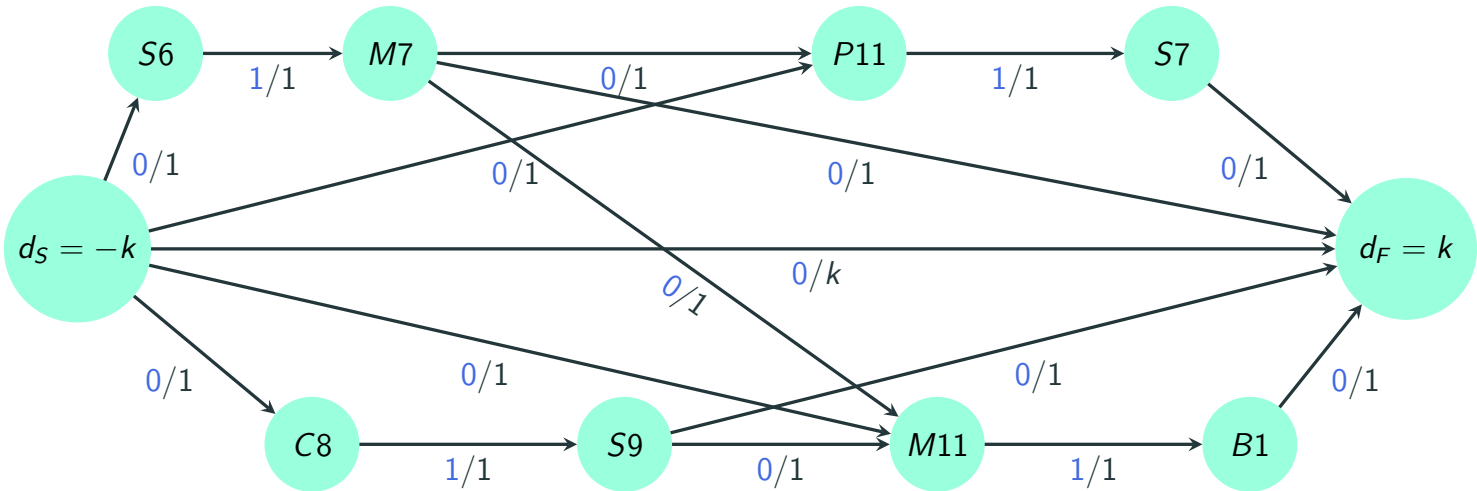
- Flight 1: SYD 6am – MEL 7am
- Flight 2: CBR 8am – SYD 9am
- Flight 3: MEL 11am – BNE 1pm (reachable from flights 1 and 2)
- Flight 4: PER 11am – SYD 7pm (reachable from flight 1)



Add an edge with lower bound 0 and capacity k between S and F (it is fine for planes to stay idle if they are not needed).

Airline scheduling

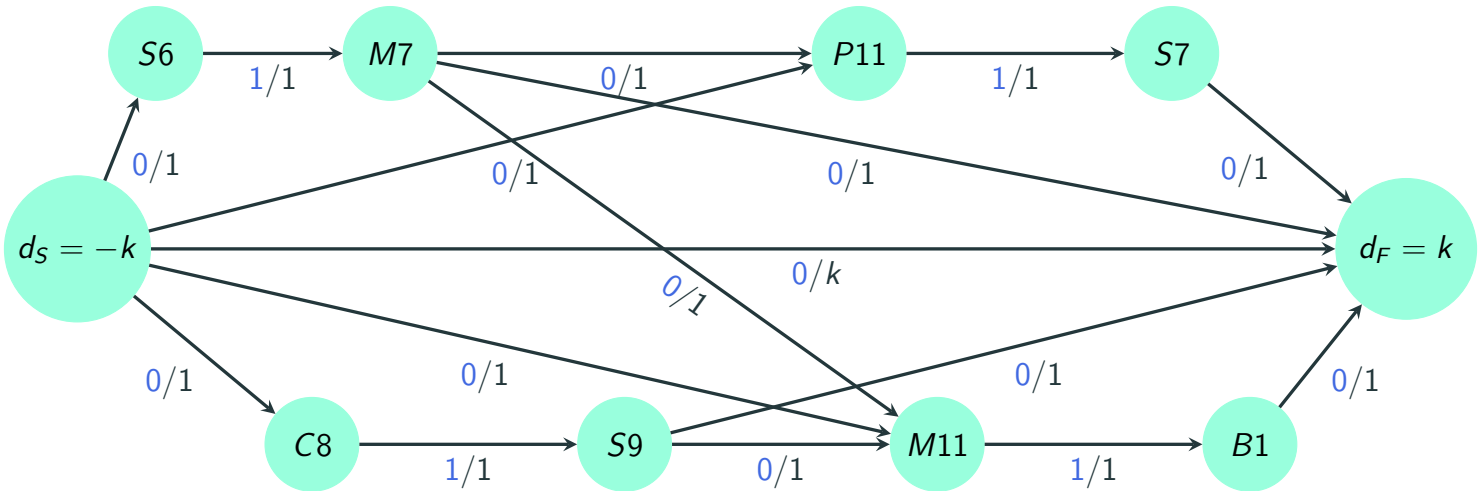
- Flight 1: SYD 6am – MEL 7am
- Flight 2: CBR 8am – SYD 9am
- Flight 3: MEL 11am – BNE 1pm (reachable from flights 1 and 2)
- Flight 4: PER 11am – SYD 7pm (reachable from flight 1)



For every reachable flight, add an edge with lower bound 0 and capacity 1 between the arrival of the previous flight, and the departure of the next one.

Airline scheduling

- Flight 1: SYD 6am – MEL 7am
- Flight 2: CBR 8am – SYD 9am
- Flight 3: MEL 11am – BNE 1pm (reachable from flights 1 and 2)
- Flight 4: PER 11am – SYD 7pm (reachable from flight 1)



Solve the circulation with demands and lower bounds problem to find if there is a feasible solution (and if there is one, the route each plane should follow).

Further examples (applied class next week)

- Project selection problem:
  - There is a set of projects with dependencies and you should decide which ones to undertake.
  - Some projects generate profits if completed, others result in a loss. You should maximise the profit.
- Elimination in sport tournaments:
  - A tournament is in progress and you know the standings and the games still to be played.
  - Determine if one team still have a chance of finishing as the leader.
- Find path covers in graphs.

Further examples

There are many other applications, such as:

- Open-pit mining
- Image segmentation (e.g., background/foreground segmentation)
- Network connectivity
- Data mining
- Distributed computing
- Network intrusion detection
- Edge-disjoint paths in graphs
- Network reliability
- Multi-camera scene reconstruction
- Gene function prediction

Reading

- Course Notes: Chapter 9
- You can also check algorithms' textbooks for contents related to this lecture, e.g.:
  - KT: Sections 7.5, 7.7, 7.8 and 7.9

Concluding remarks

- Take home message:
  - Network Flow can be applied to solve many non-trivial combinatorial problems.
  - Circulation with demands and lower bounds can be very useful in those applications.
- Things to do:
  - **Make sure you understand circulation with demands and lower bounds.**
  - **Make sure you understand how to apply the network flow techniques to solve combinatorial problems.**
- Coming up next: Retrieval data structures for strings.