香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

Flow and Graph Matching (I)

Ethen Yuen {ethening}

2026-04-18

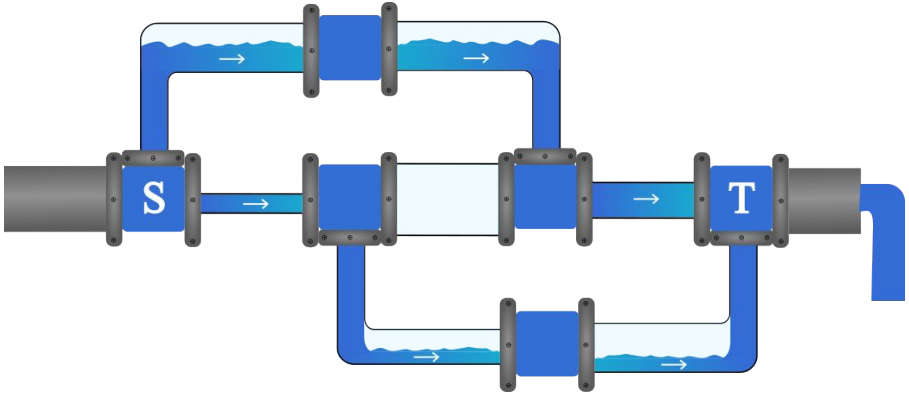
Agenda

- Maximum Flow
  - Ford-Fulkerson Algorithm
  - Modeling Technique
- Minimum Cut
  - Max-Flow Min-Cut Theorem
  - Set Partitioning Model and “Optimistic Thinking”
- *Team* Problem Set

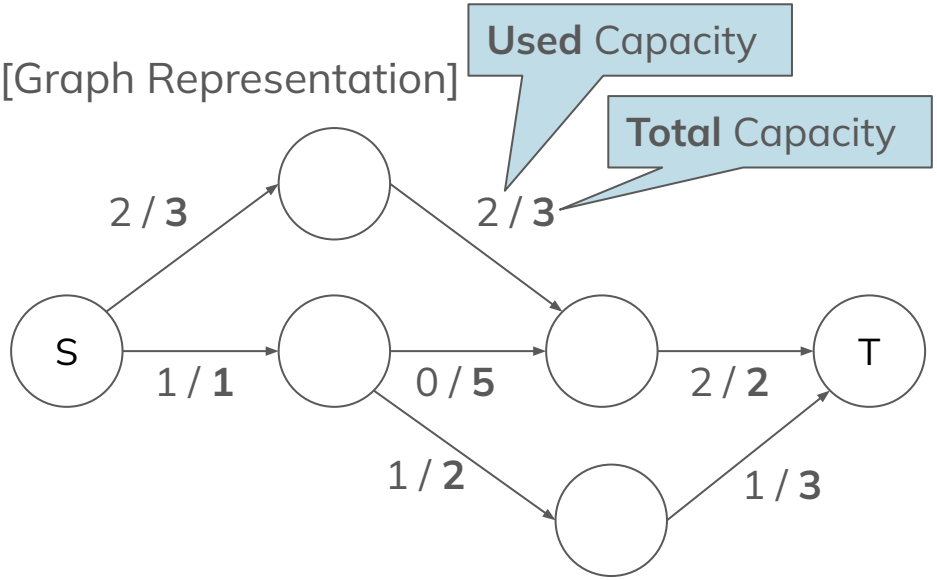
Maximum Flow

Maximum Flow Problem

[Flow Network in Real Life]

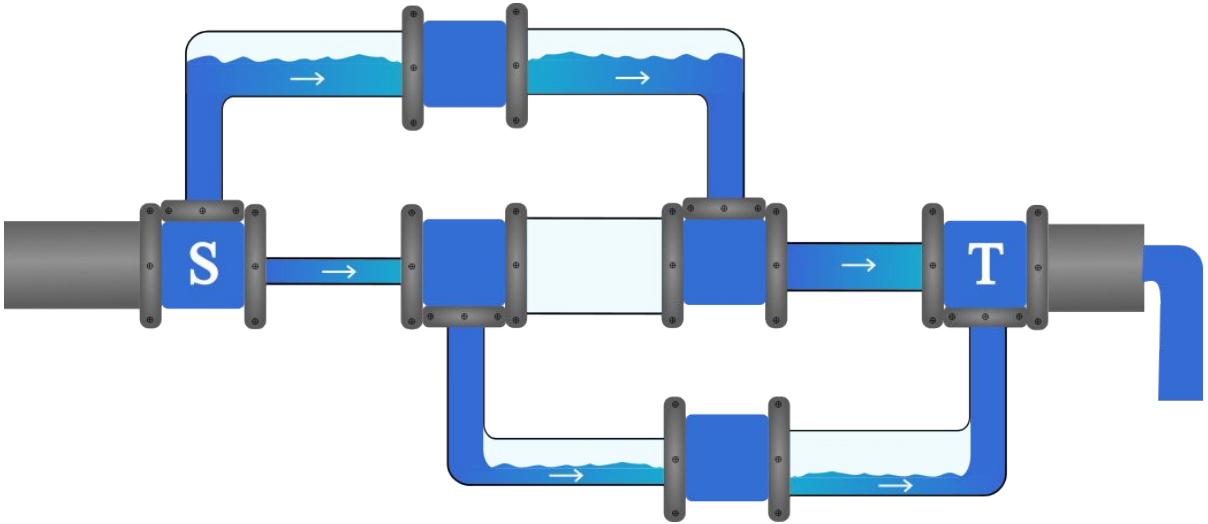


[Graph Representation]



Maximum Flow Problem

There are some nodes in a flow network, connected with **unidirectional** pipes. Each pipe has a **maximum capacity**. Find the maximum amount of flow we can send from the **source S** to the **sink T** per unit time.



Maximum Flow Problem

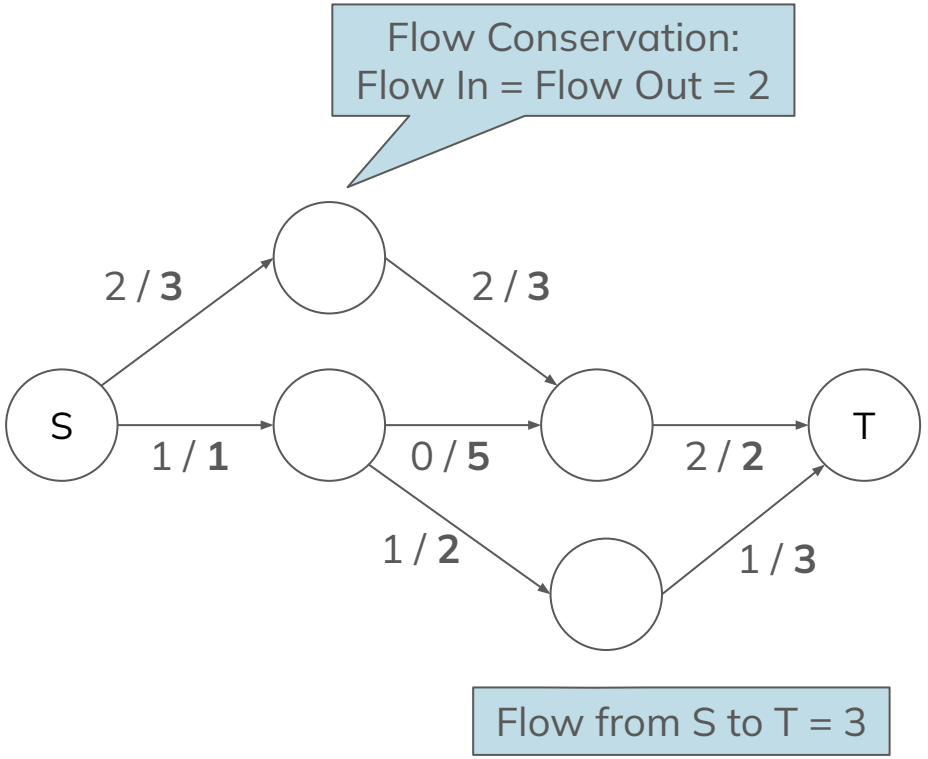
Flow Conservation: For all vertices except S and T, flow in = flow out.

We aim to maximize the **flow from S to T**:

(flow out for S) - (flow in for S)

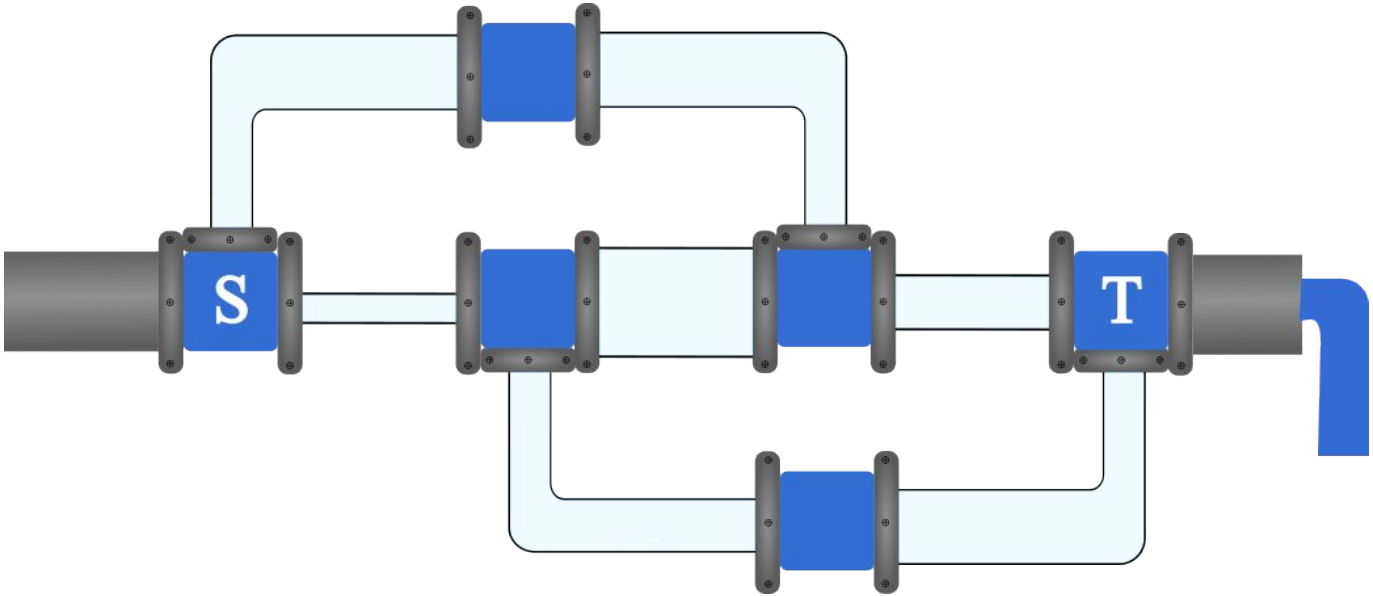
Or equivalently:

(flow in for T) - (flow out for T)



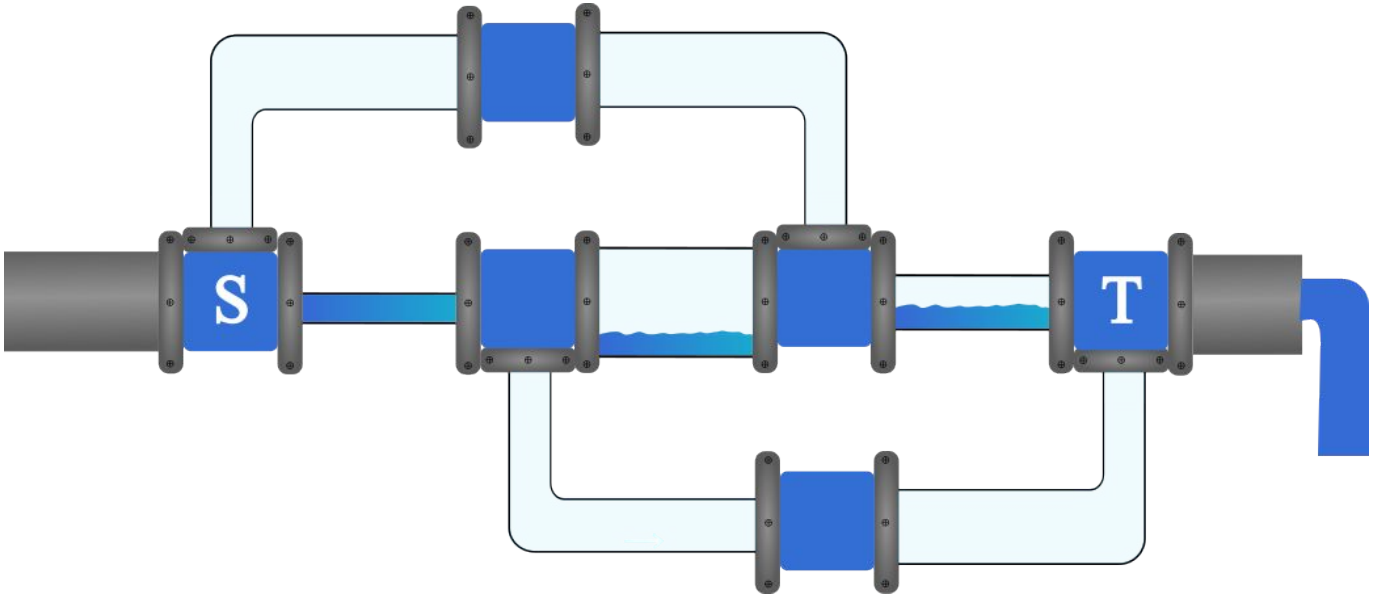
Ford–Fulkerson Algorithm – Intuition

You are asked to solve this problem in real life. What would you do?



Ford–Fulkerson Algorithm – Intuition

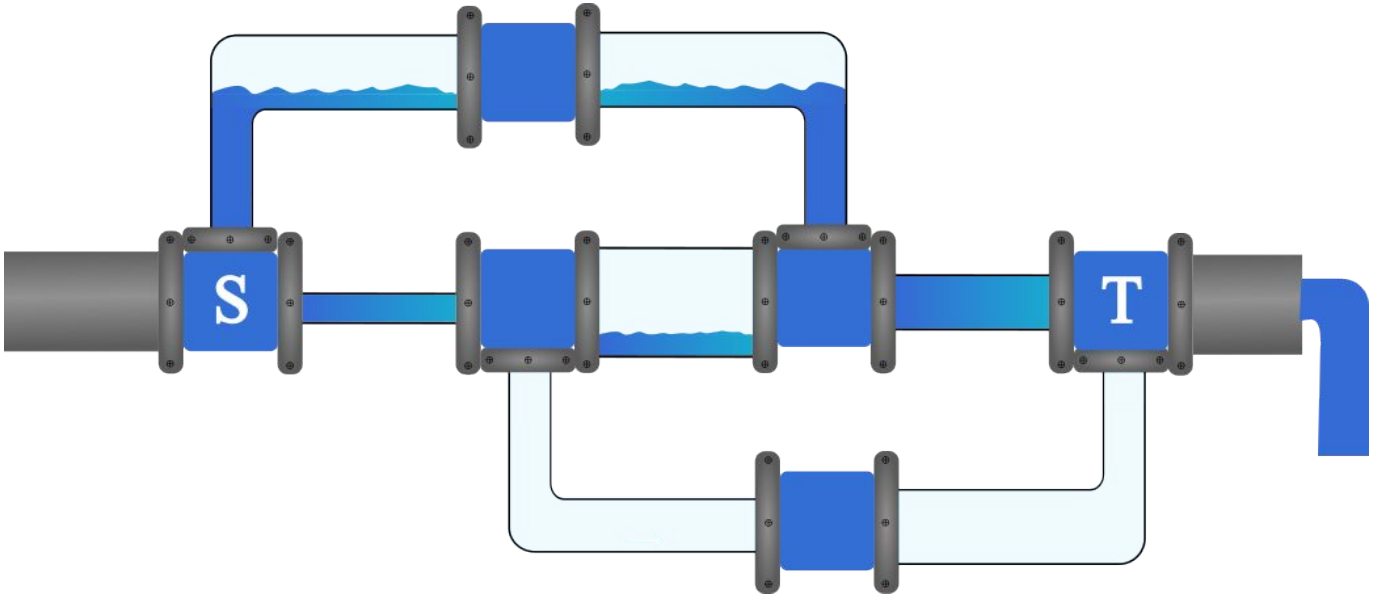
You are asked to solve this problem in real life. What would you do?



1. Turn on the tap

Ford–Fulkerson Algorithm – Intuition

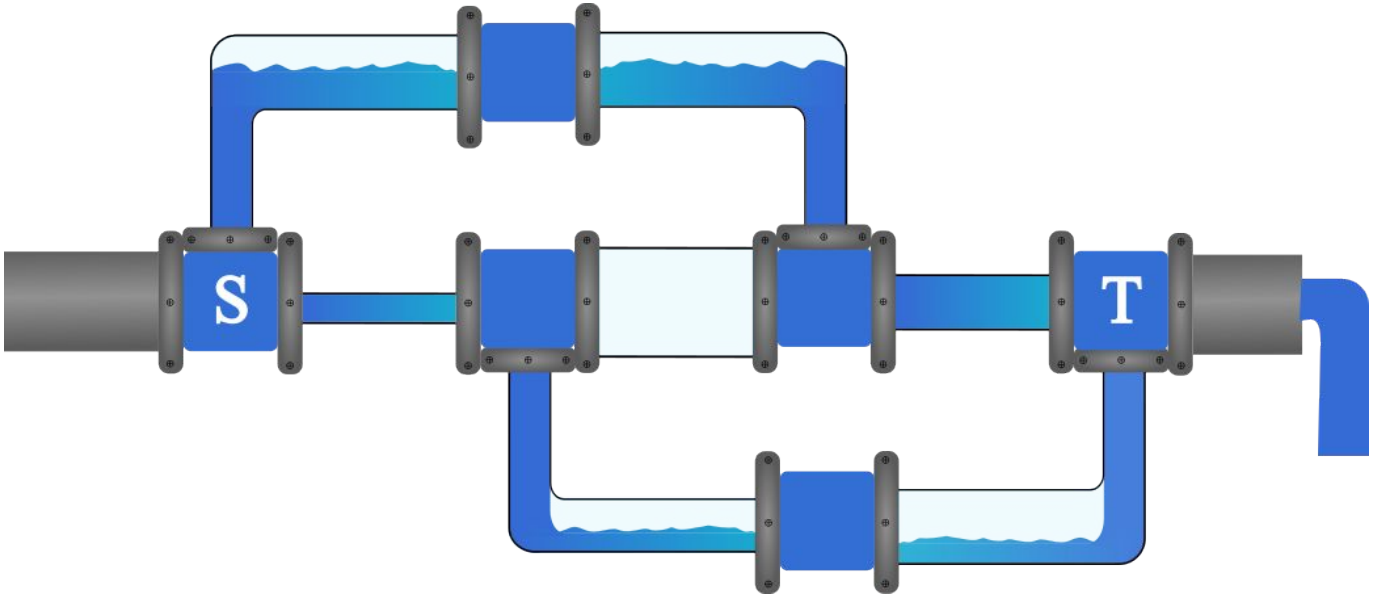
You are asked to solve this problem in real life. What would you do?



1. Turn on the tap
2. A bit more...

Ford–Fulkerson Algorithm – Intuition

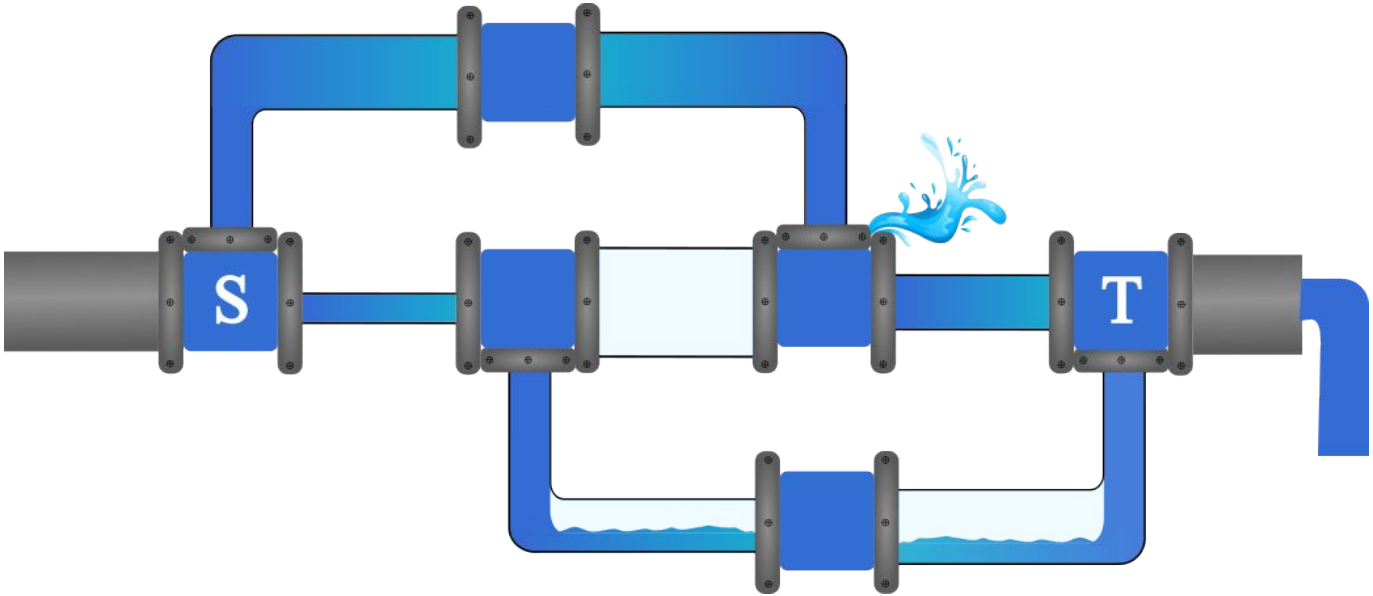
You are asked to solve this problem in real life. What would you do?



1. Turn on the tap
2. A bit more...
3. A bit more...

Ford–Fulkerson Algorithm – Intuition

You are asked to solve this problem in real life. What would you do?



1. Turn on the tap
2. A bit more...
3. A bit more...

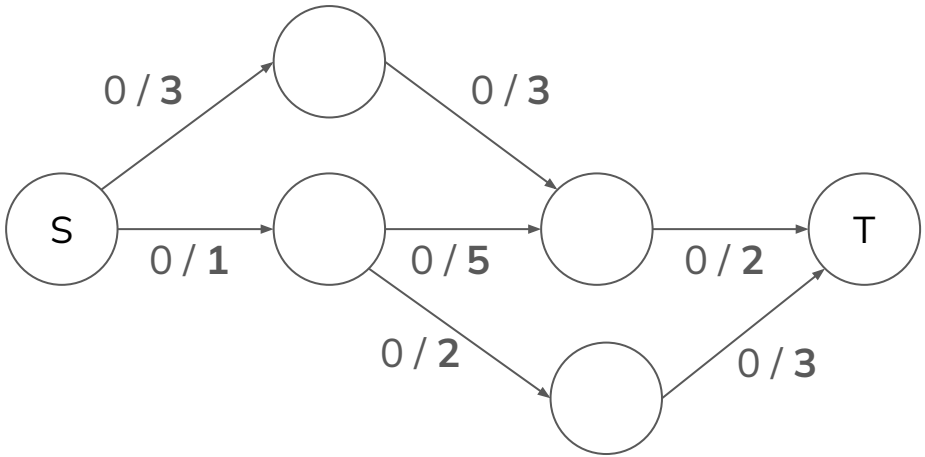
↑ Maximum Flow

4. A bit more...

No more flow to T

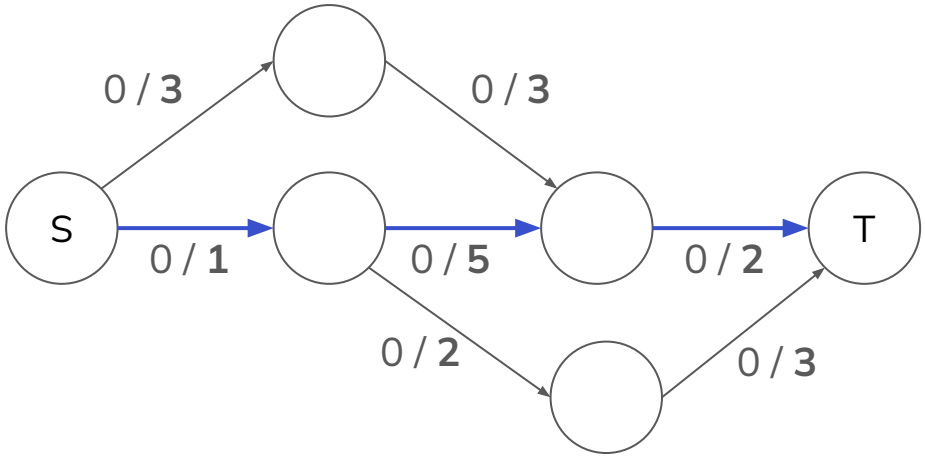
Ford–Fulkerson Algorithm – Intuition

```
REPEAT {
    Find a path from S to T
    If (a path is found)
        Record the used capacity
        Increment maximum flow by 1
    Else
        Break out of loop
}
```



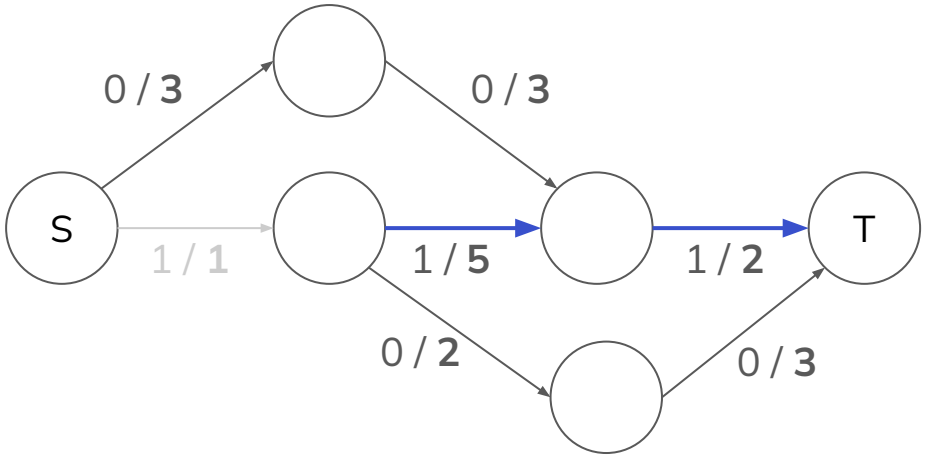
Ford–Fulkerson Algorithm – Intuition

```
REPEAT {
    Find a path from S to T
    If (a path is found)
        Record the used capacity
        Increment maximum flow by 1
    Else
        Break out of loop
}
```



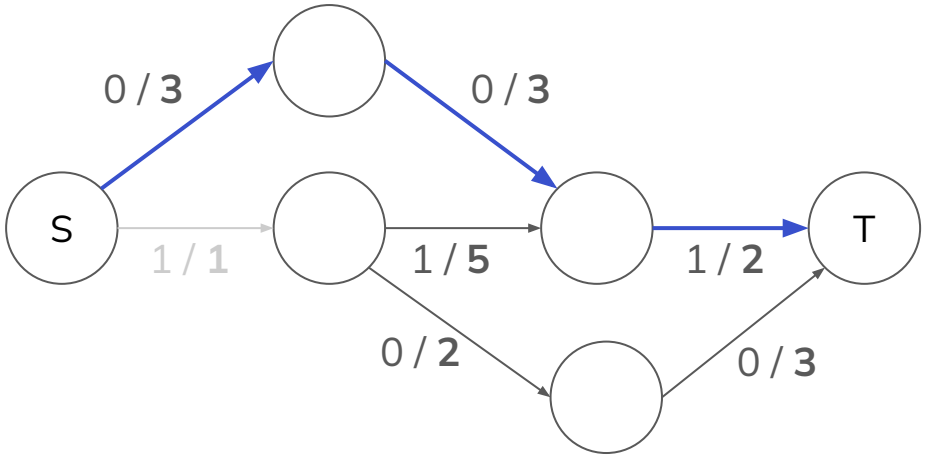
Ford–Fulkerson Algorithm – Intuition

```
REPEAT {
    Find a path from S to T
    If (a path is found)
        Record the used capacity
        Increment maximum flow by 1
    Else
        Break out of loop
}
```



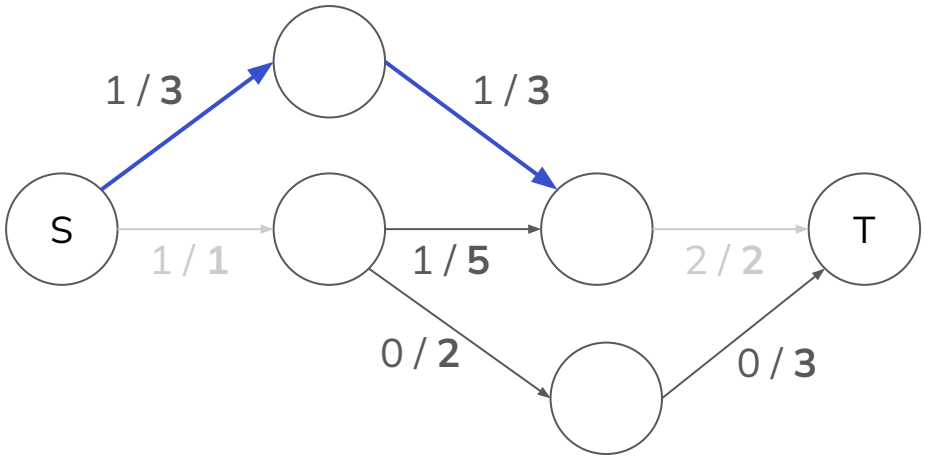
Ford–Fulkerson Algorithm – Intuition

```
REPEAT {
    Find a path from S to T
    If (a path is found)
        Record the used capacity
        Increment maximum flow by 1
    Else
        Break out of loop
}
```



Ford–Fulkerson Algorithm – Intuition

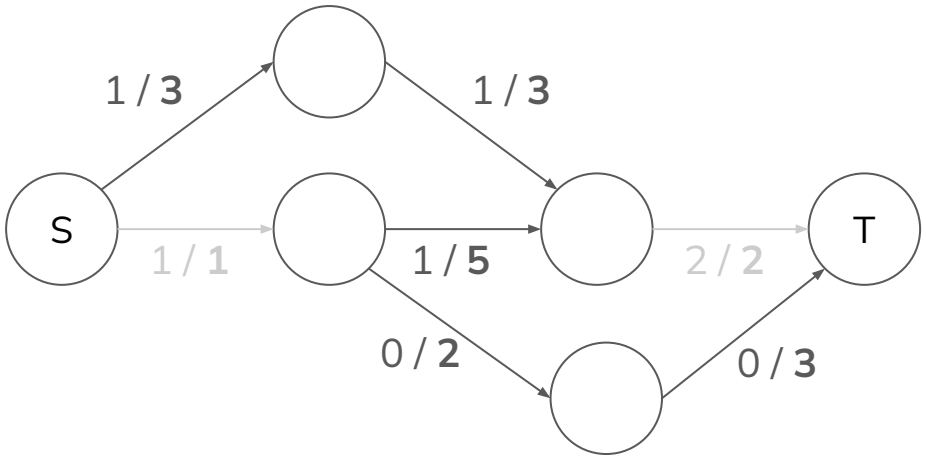
```
REPEAT {
    Find a path from S to T
    If (a path is found)
        Record the used capacity
        Increment maximum flow by 1
    Else
        Break out of loop
}
```



Ford–Fulkerson Algorithm – Intuition

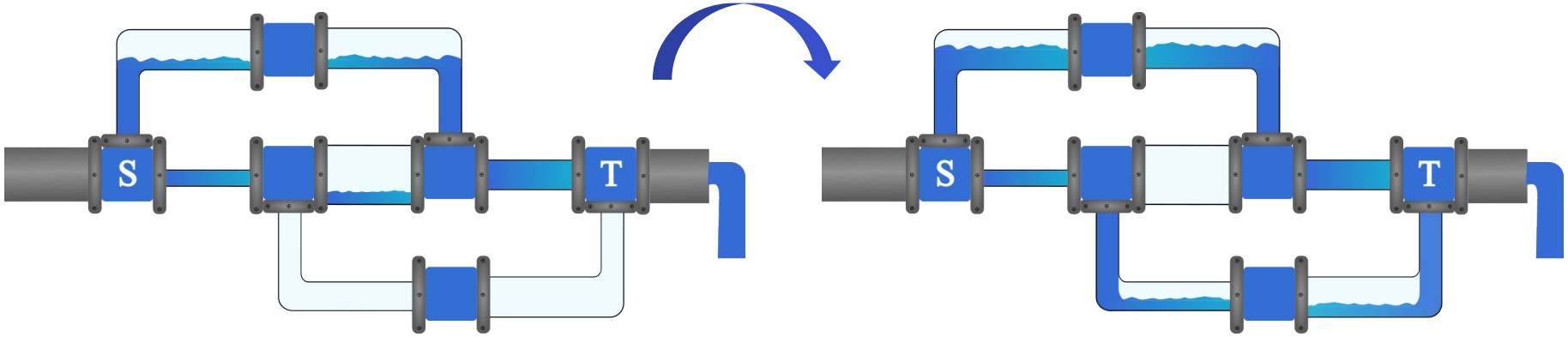
```
REPEAT {
    Find a path from S to T
    If (a path is found)
        Record the used capacity
        Increment maximum flow by 1
    Else
        Break out of loop
}
```

No more paths
Max Flow = 2?



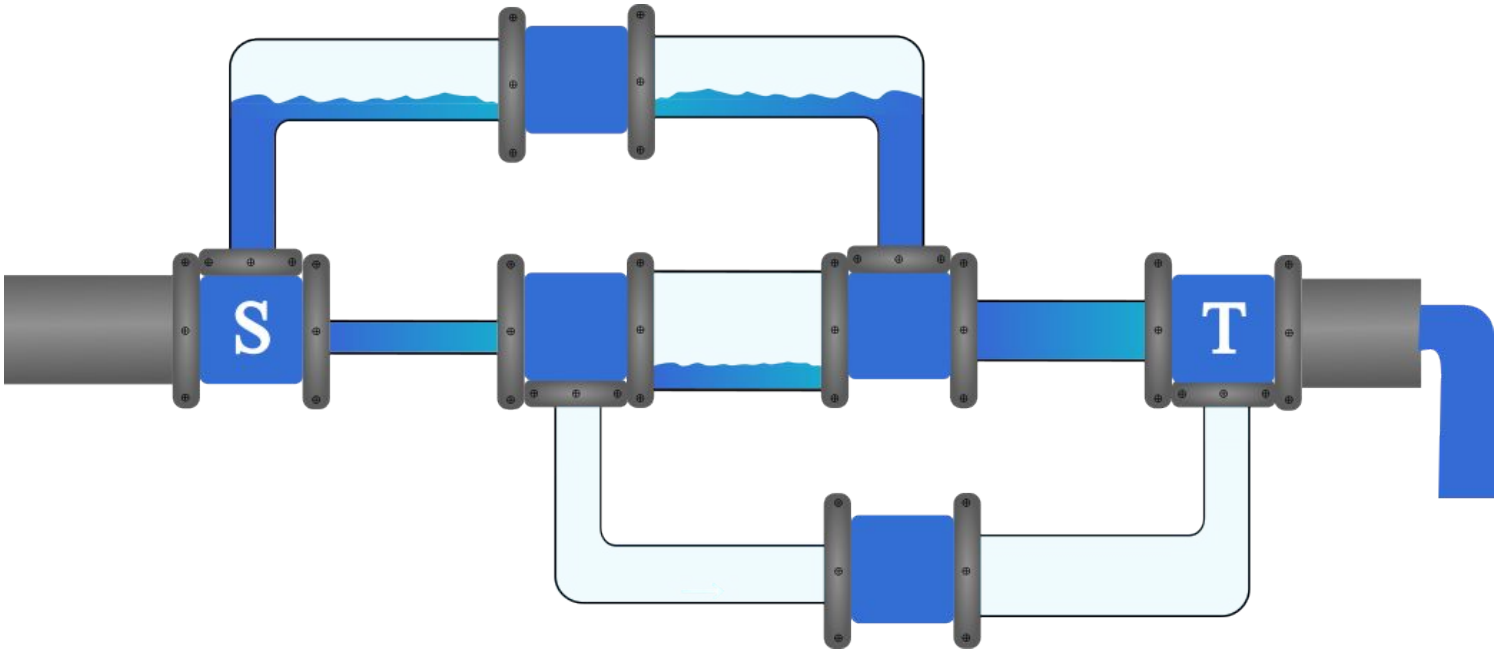
Ford–Fulkerson Algorithm – Intuition

Notice that in our “real life” example, the water will “magically” flow correctly when the flow increases from 2 to 3:



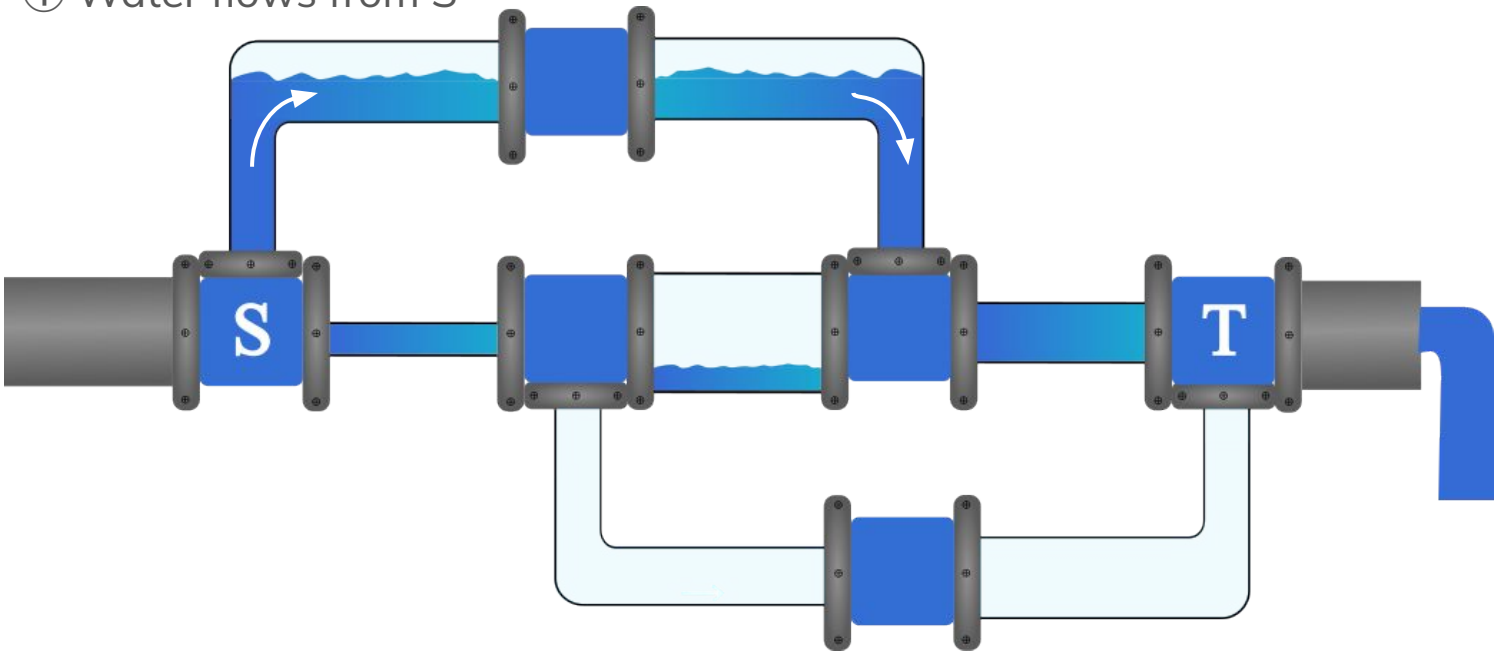
How did that happen?

Ford–Fulkerson Algorithm – Intuition

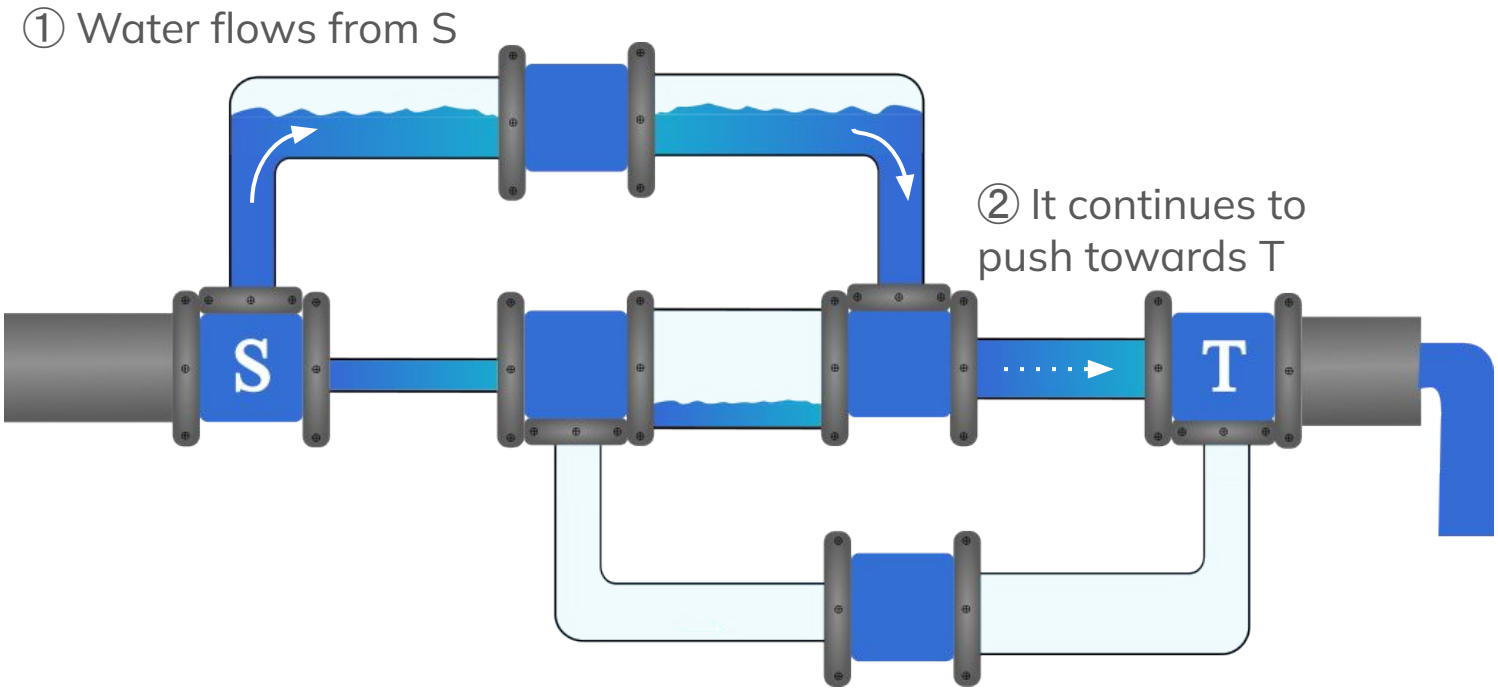


Ford–Fulkerson Algorithm – Intuition

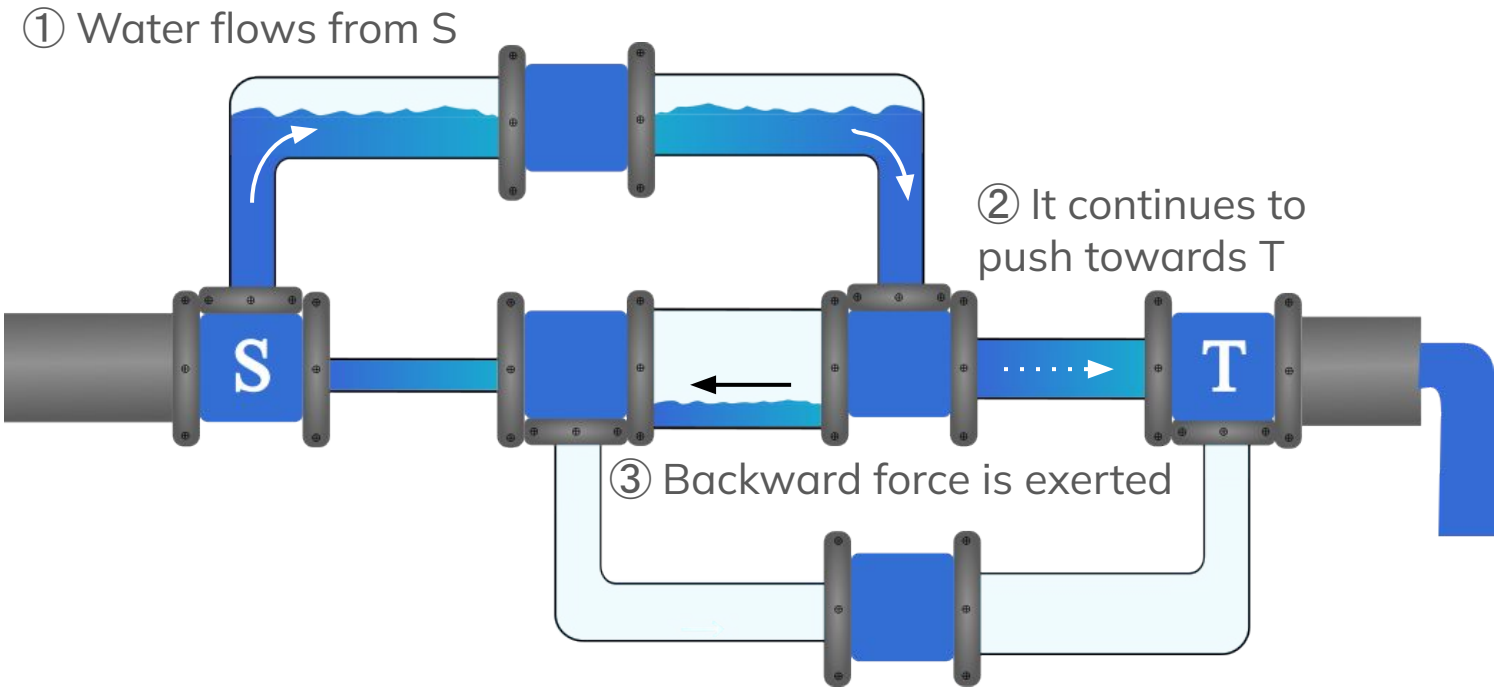
① Water flows from S



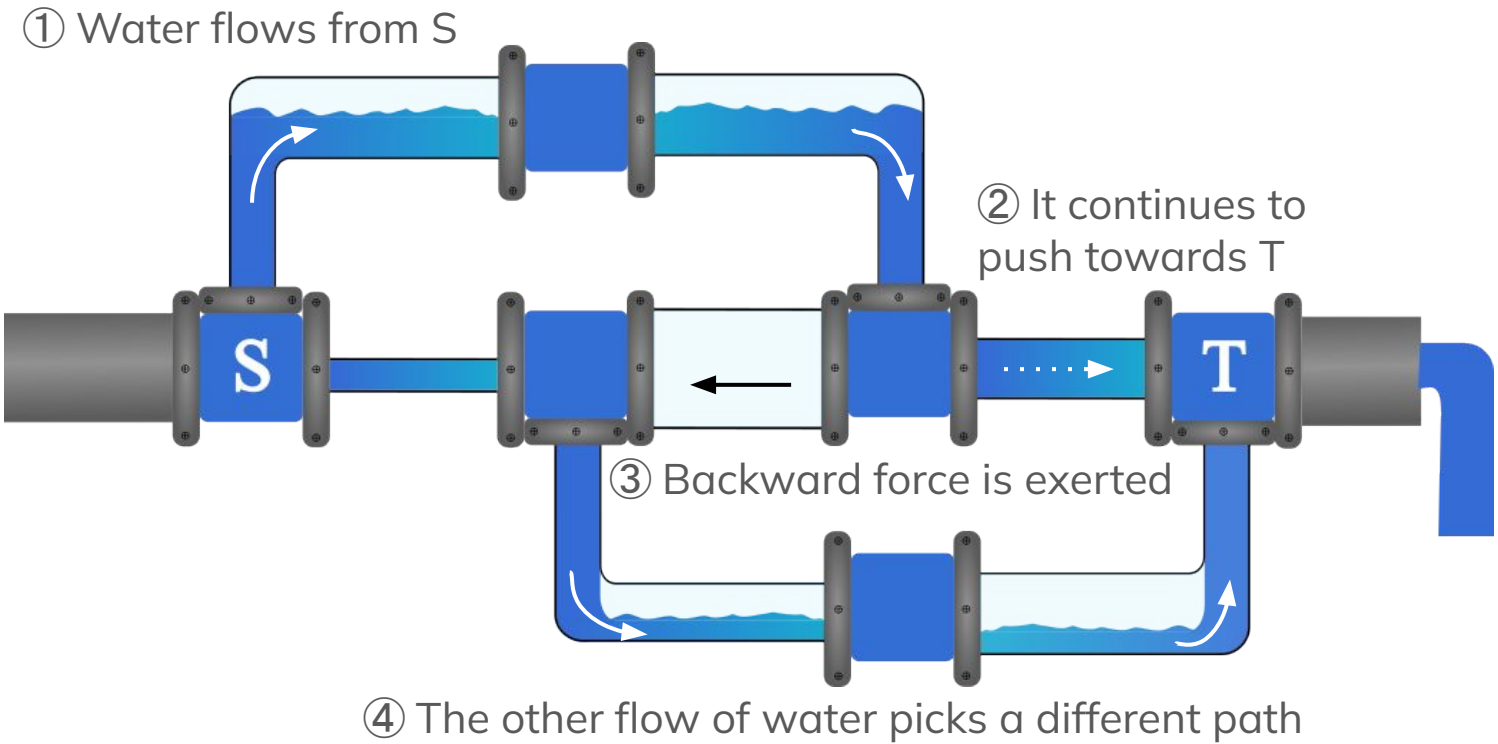
Ford–Fulkerson Algorithm – Intuition



Ford–Fulkerson Algorithm – Intuition



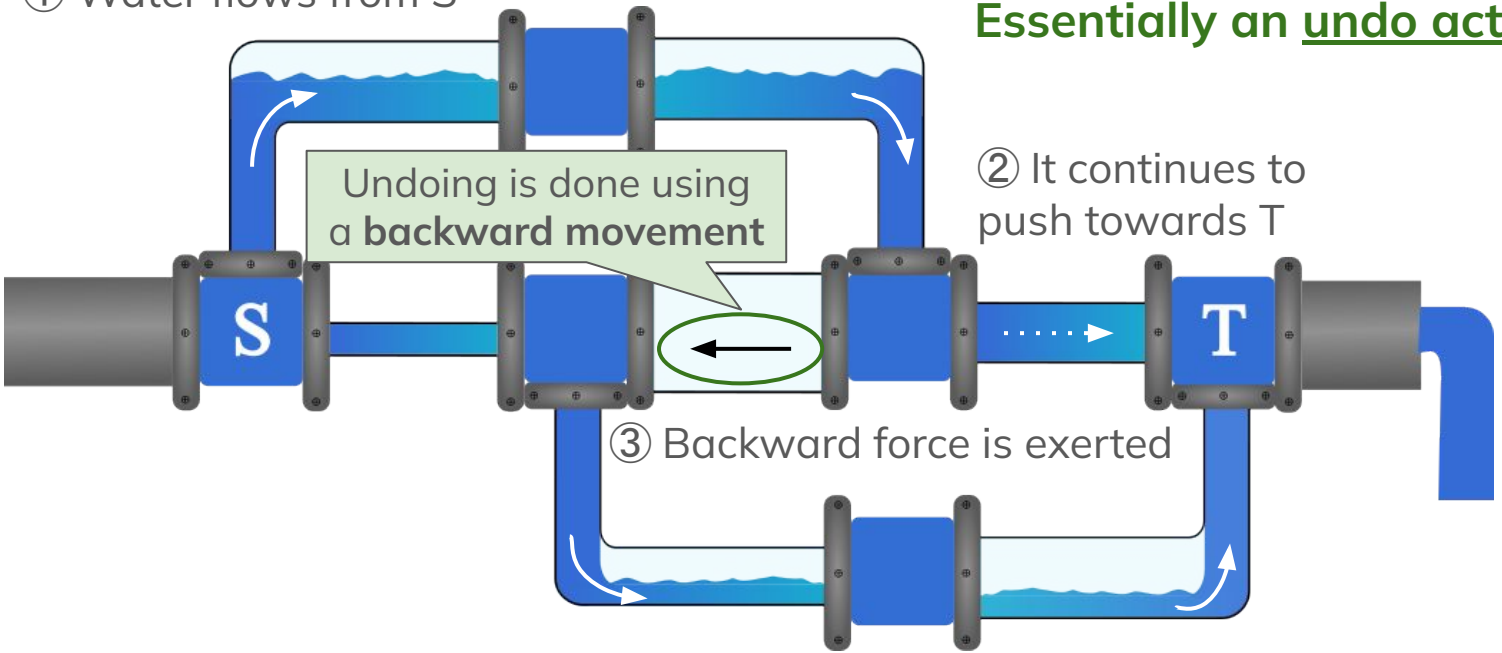
Ford–Fulkerson Algorithm – Intuition



Ford–Fulkerson Algorithm – Intuition

① Water flows from S

Essentially an undo action!



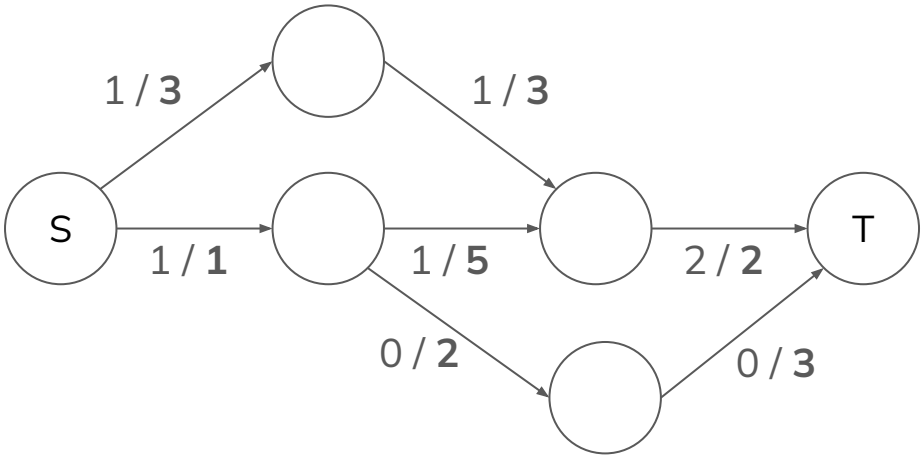
④ The other flow of water picks a different path

Ford–Fulkerson Algorithm – Implementation

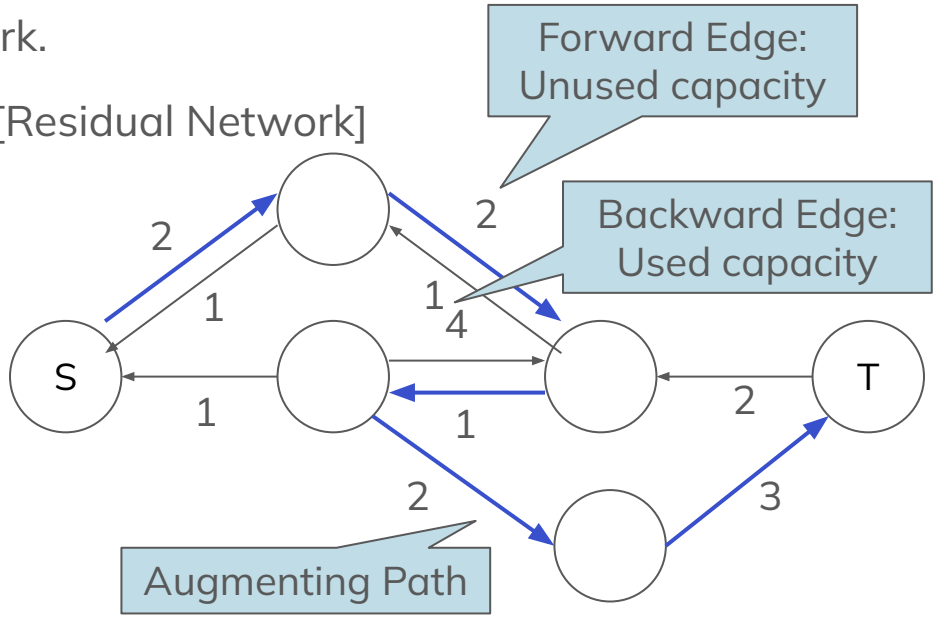
Residual Network: The current state of the flow network together with the backward edges for “undoing” actions.

Augmenting Path: Path on the residual network.

[Flow Network]

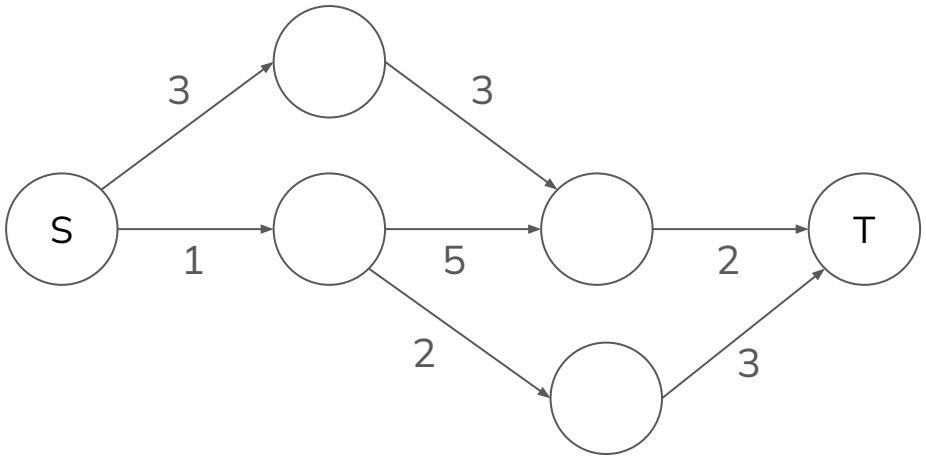


[Residual Network]



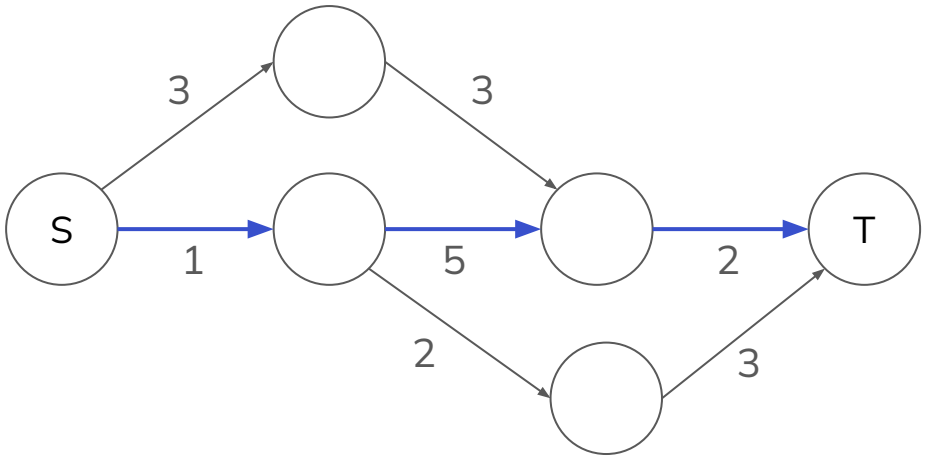
Ford–Fulkerson Algorithm – Implementation

```
REPEAT {
    Find an augmenting path from S to T
    If (a path is found)
        Update the residual network
        Increment maximum flow by 1
    Else
        Break out of loop
}
```



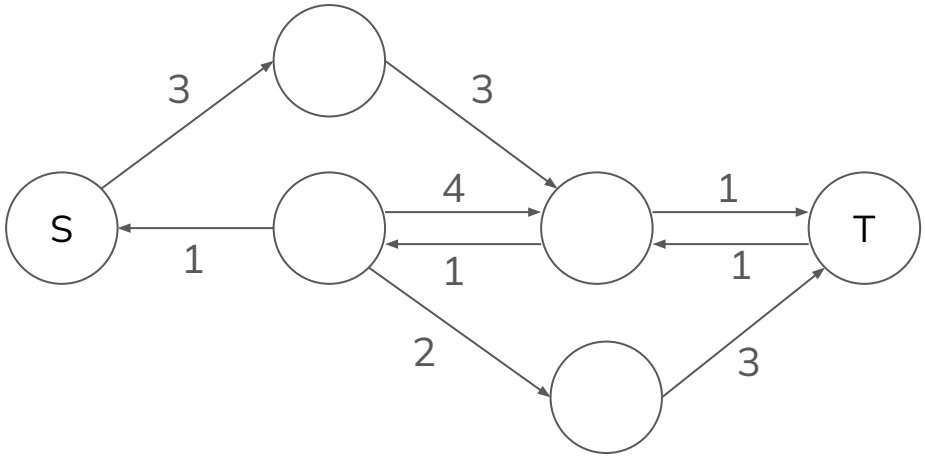
Ford–Fulkerson Algorithm – Implementation

```
REPEAT {
    Find an augmenting path from S to T
    If (a path is found)
        Update the residual network
        Increment maximum flow by 1
    Else
        Break out of loop
}
```



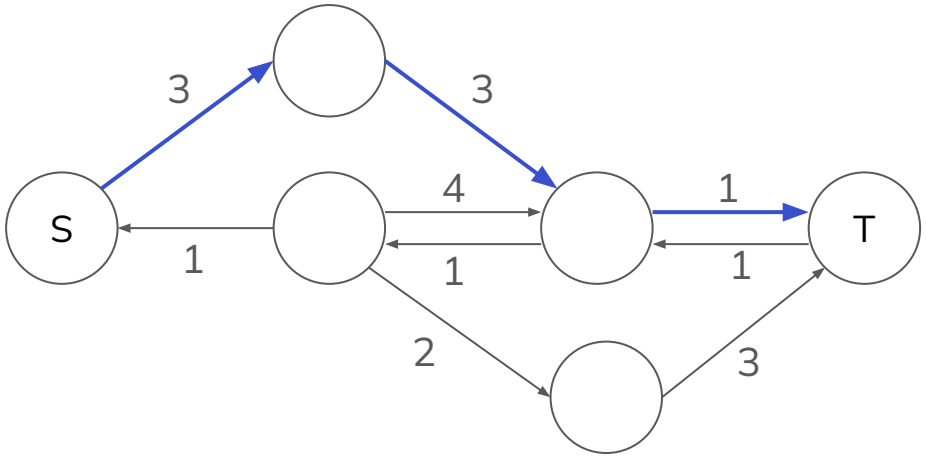
Ford–Fulkerson Algorithm – Implementation

```
REPEAT {
    Find an augmenting path from S to T
    If (a path is found)
        Update the residual network
        Increment maximum flow by 1
    Else
        Break out of loop
}
```



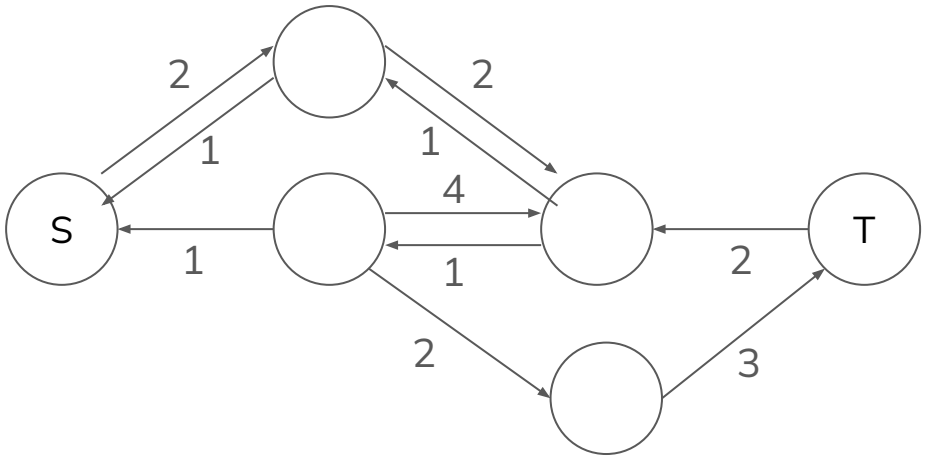
Ford–Fulkerson Algorithm – Implementation

```
REPEAT {
    Find an augmenting path from S to T
    If (a path is found)
        Update the residual network
        Increment maximum flow by 1
    Else
        Break out of loop
}
```



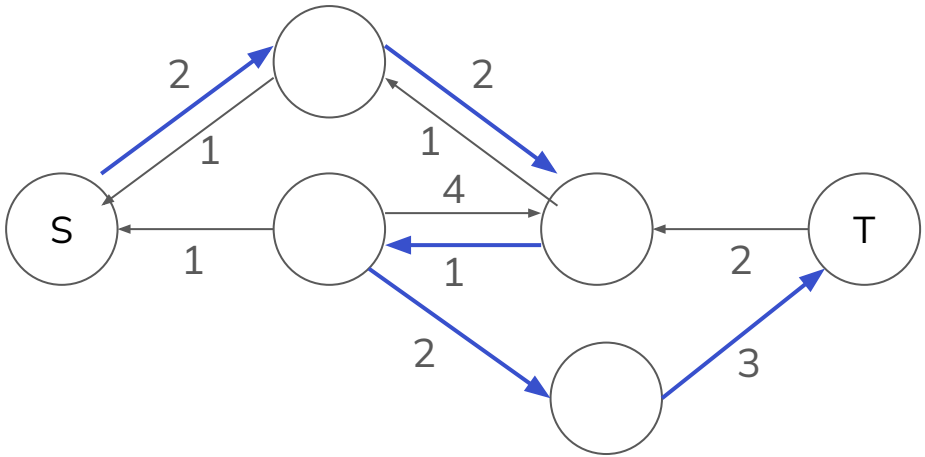
Ford–Fulkerson Algorithm – Implementation

```
REPEAT {
    Find an augmenting path from S to T
    If (a path is found)
        Update the residual network
        Increment maximum flow by 1
    Else
        Break out of loop
}
```



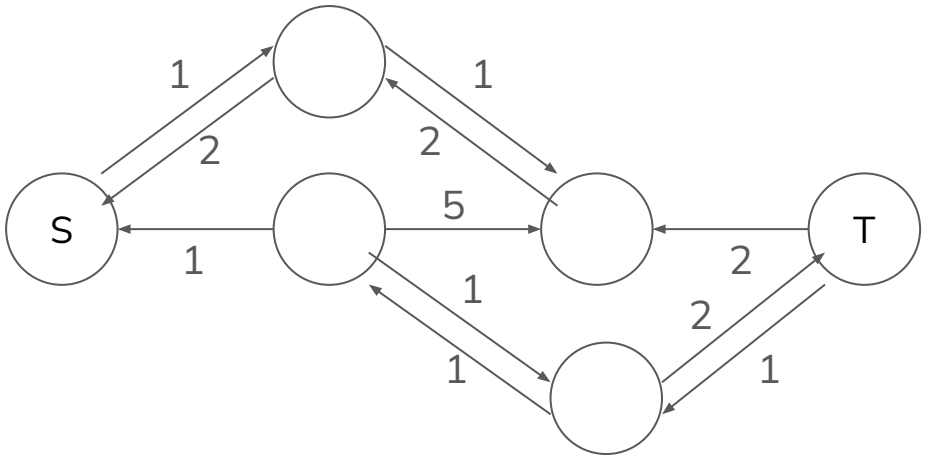
Ford–Fulkerson Algorithm – Implementation

```
REPEAT {
    Find an augmenting path from S to T
    If (a path is found)
        Update the residual network
        Increment maximum flow by 1
    Else
        Break out of loop
}
```



Ford–Fulkerson Algorithm – Implementation

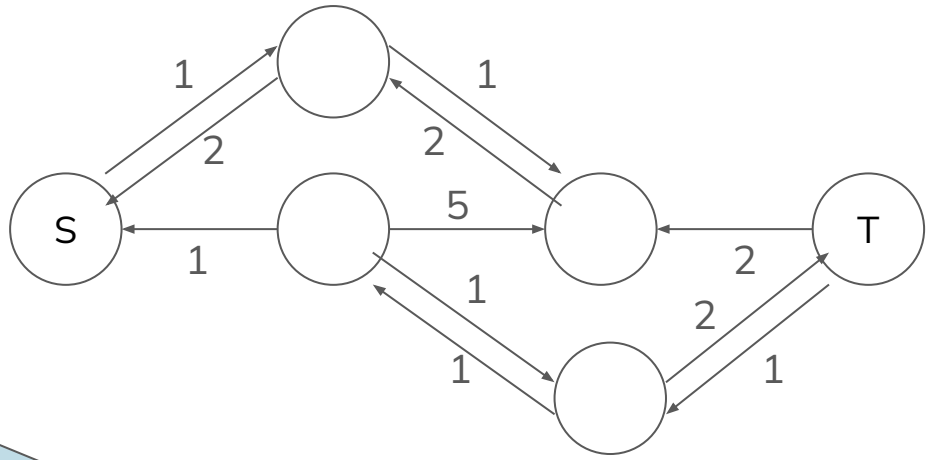
```
REPEAT {
    Find an augmenting path from S to T
    If (a path is found)
        Update the residual network
        Increment maximum flow by 1
    Else
        Break out of loop
}
```



Ford–Fulkerson Algorithm – Implementation

```
REPEAT {
    Find an augmenting path from S to T
    If (a path is found)
        Update the residual network
        Increment maximum flow by 1
    Else
        Break out of loop
}
```

No more paths
Max Flow = 3



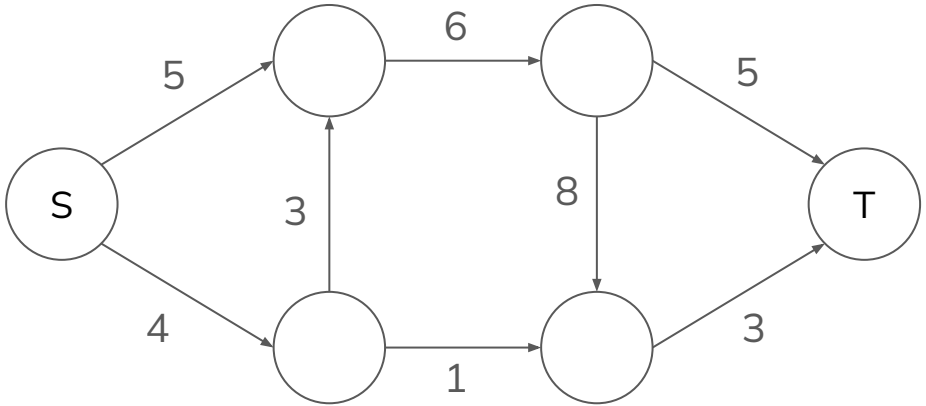
Time Complexity: **O(EF)**

Number of edges

Maximum flow

Quick Check

🔤 Consider the following flow network. Find the maximum flow from S to T.



(Answer: 7)

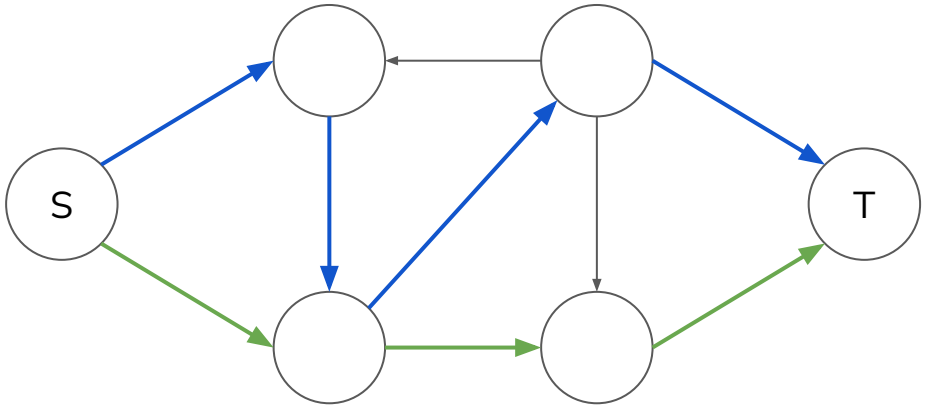
Other Maximum Flow Algorithms

- There exists other faster flow algorithms (not within the scope of IOI).

Year	Method	Time Complexity
1955	Ford–Fulkerson algorithm	O(EF)
1970	Dinic's algorithm	$O(V^2E)$
1983	Dinic's algorithm + dynamic trees	O(VE log V)
1988	Push–relabel algorithm + dynamic trees	$O(VE \log(V^2/E))$
…		
2022	Chen, Kyng, Liu, Peng, Gutenberg and Sachdeva's algorithm	$O(E^{1+o(1)} \log F)$ *** Almost linear**

Solving Graph Problems with Flow

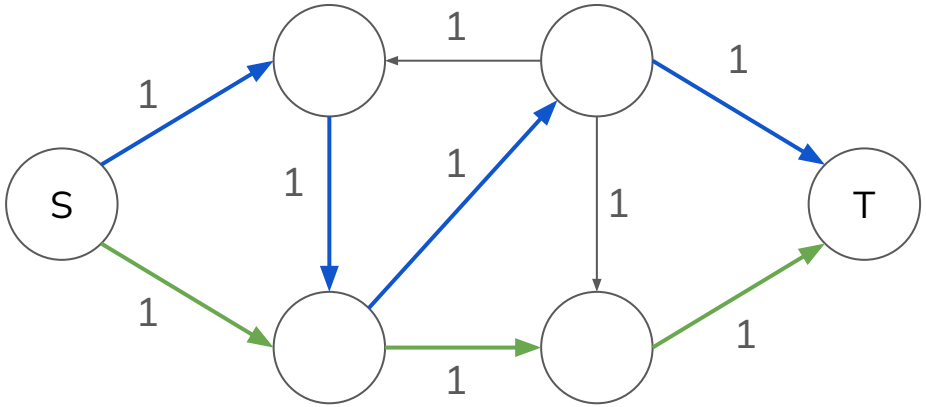
Edge-disjoint paths: Find the maximum number of **edge-disjoint paths** from the **source** to the **sink**, i.e. each edge appears in **at most one path**.



Solving Graph Problems with Flow

Edge-disjoint paths: Find the maximum number of **edge-disjoint paths** from the **source** to the **sink**, i.e. each edge appears in **at most one path**.

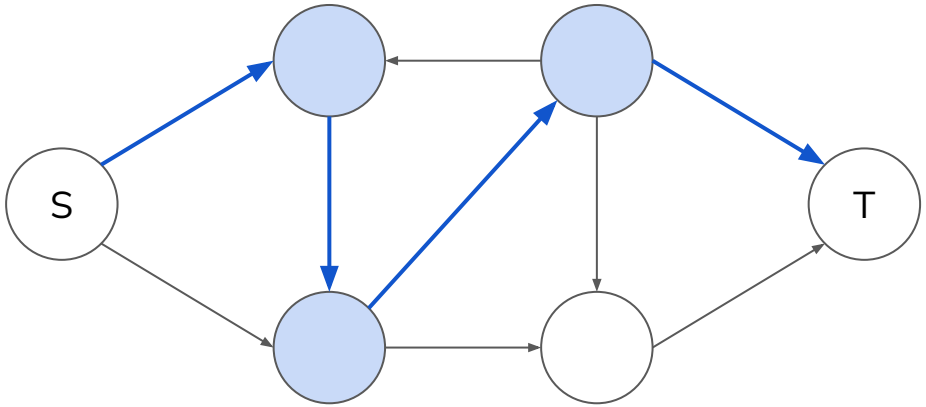
- Essentially the maximum flow from S to T!



Solving Graph Problems with Flow

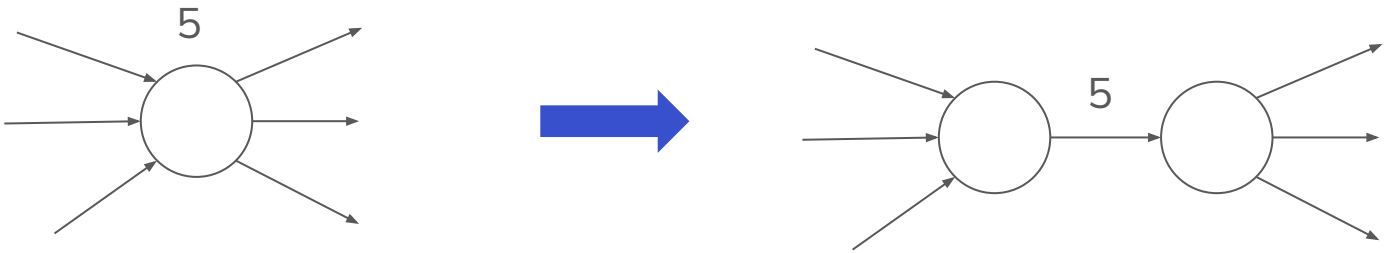
Vertex-disjoint paths: Find the maximum number of **vertex-disjoint paths** from the source to the sink, i.e. each **vertex (except S and T)** appears in at most one path.

- **Issue**: Flow only supports edge capacities but not vertex capacities.



Vertex Splitting Technique

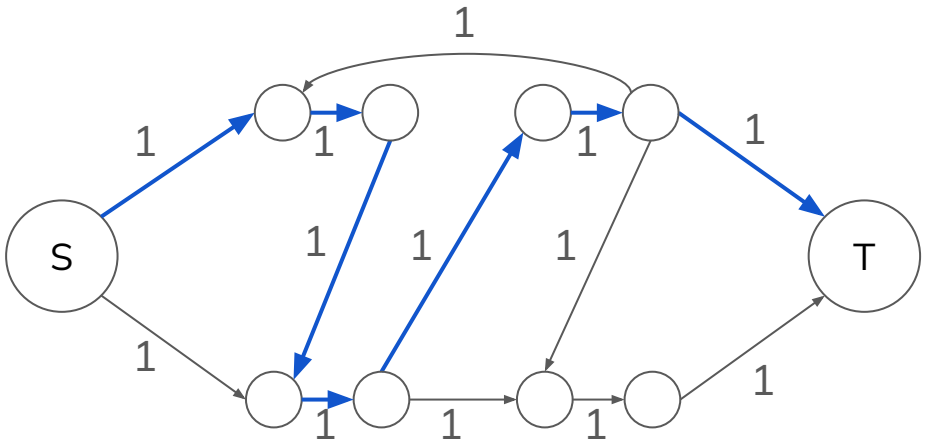
- Split a vertex into two vertices connected with an edge.
- By doing so, we can use the capacity of the edge to control the flow via the vertex!



Solving Graph Problems with Flow

Vertex-disjoint paths: Find the maximum number of **vertex-disjoint paths** from the source to the sink, i.e. each **vertex (except S and T)** appears in at most one path.

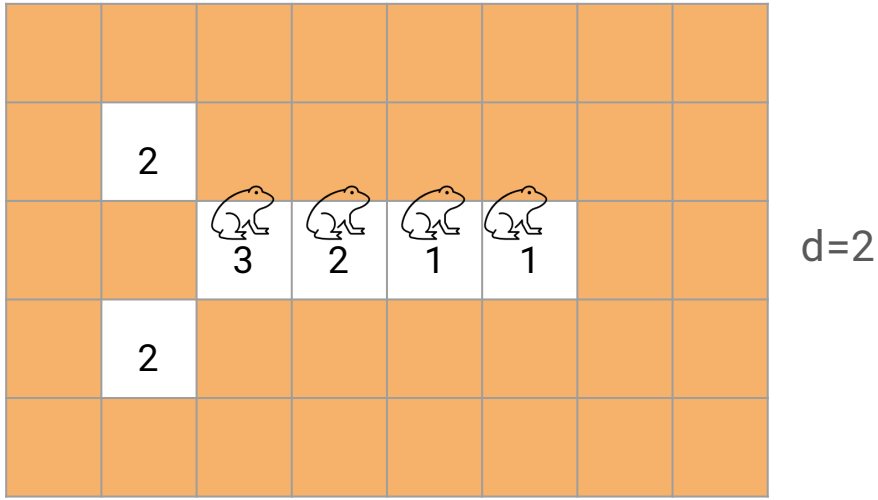
- Use the **vertex splitting technique** to support vertex capacities.



Problem Source: Luogu P2472

Modeling Non-Graph Problems with Flow

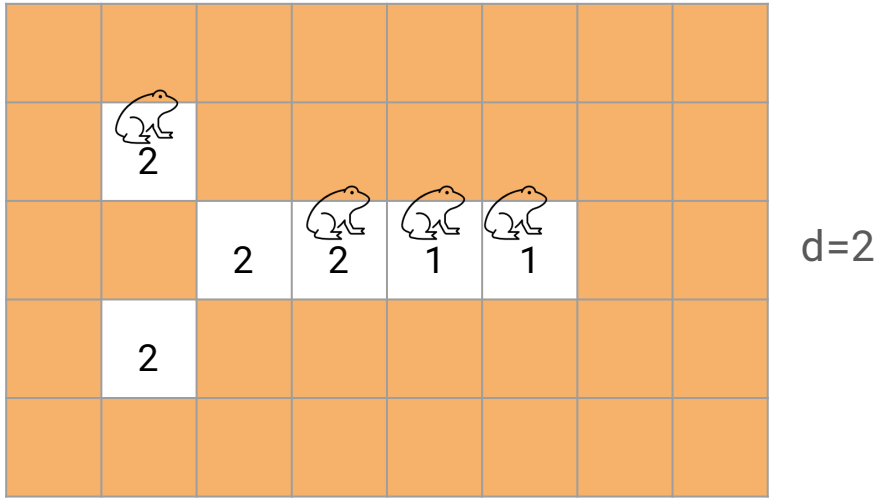
Some frogs are currently in a 2D grid. A frog can jump from its current location to another cell **no more than d units away** (in Euclidean distance). However, leaving a cell causes the cell's height to decrease by 1. A frog cannot jump to a cell with height 0. What is the maximum number of frogs that can jump outside the grid boundaries?



Problem Source: Luogu P2472

Modeling Non-Graph Problems with Flow

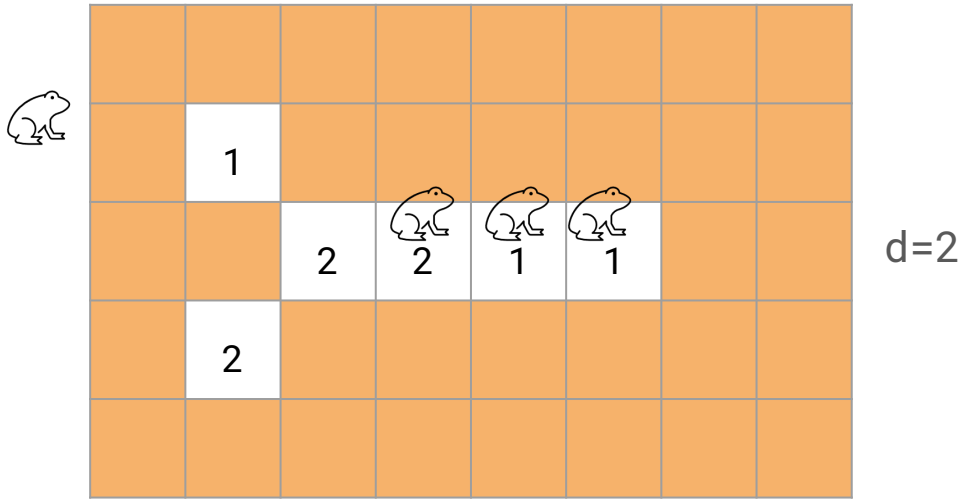
Some frogs are currently in a 2D grid. A frog can jump from its current location to another cell **no more than d units away** (in Euclidean distance). However, leaving a cell causes the cell's height to decrease by 1. A frog cannot jump to a cell with height 0. What is the maximum number of frogs that can jump outside the grid boundaries?



Problem Source: Luogu P2472

Modeling Non-Graph Problems with Flow

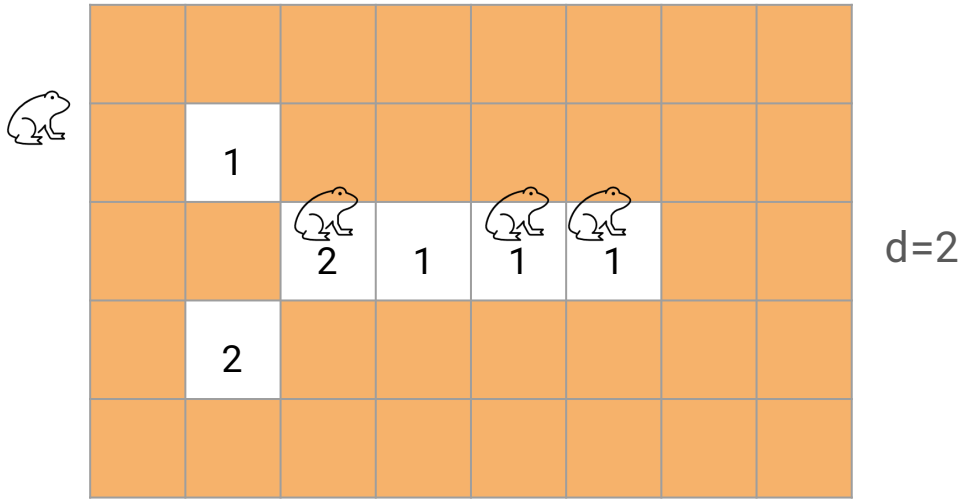
Some frogs are currently in a 2D grid. A frog can jump from its current location to another cell **no more than d units away** (in Euclidean distance). However, leaving a cell causes the cell's height to decrease by 1. A frog cannot jump to a cell with height 0. What is the maximum number of frogs that can jump outside the grid boundaries?



Problem Source: Luogu P2472

Modeling Non-Graph Problems with Flow

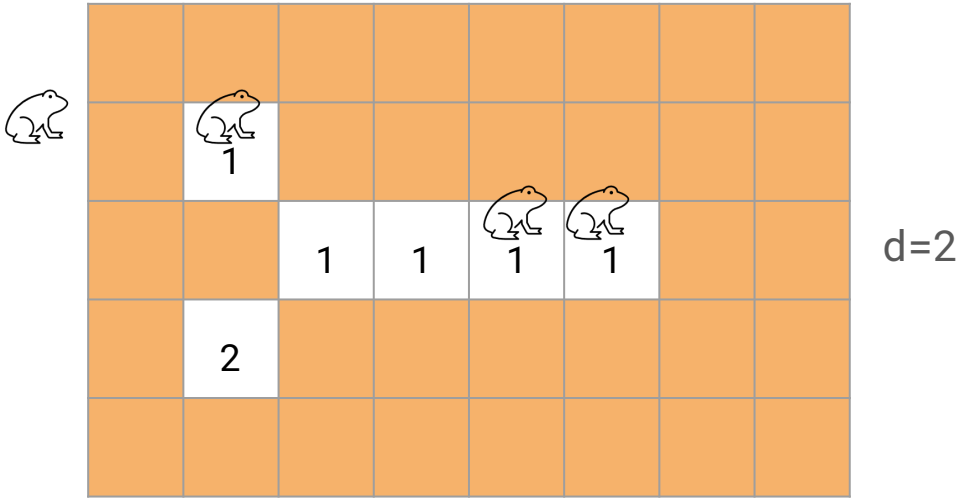
Some frogs are currently in a 2D grid. A frog can jump from its current location to another cell **no more than d units away** (in Euclidean distance). However, leaving a cell causes the cell's height to decrease by 1. A frog cannot jump to a cell with height 0. What is the maximum number of frogs that can jump outside the grid boundaries?



Problem Source: Luogu P2472

Modeling Non-Graph Problems with Flow

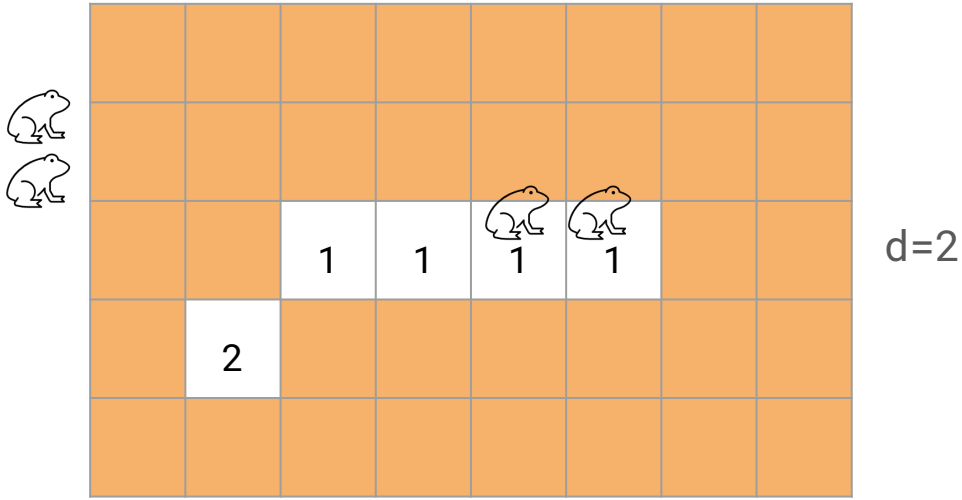
Some frogs are currently in a 2D grid. A frog can jump from its current location to another cell **no more than d units away** (in Euclidean distance). However, leaving a cell causes the cell's height to decrease by 1. A frog cannot jump to a cell with height 0. What is the maximum number of frogs that can jump outside the grid boundaries?



Problem Source: Luogu P2472

Modeling Non-Graph Problems with Flow

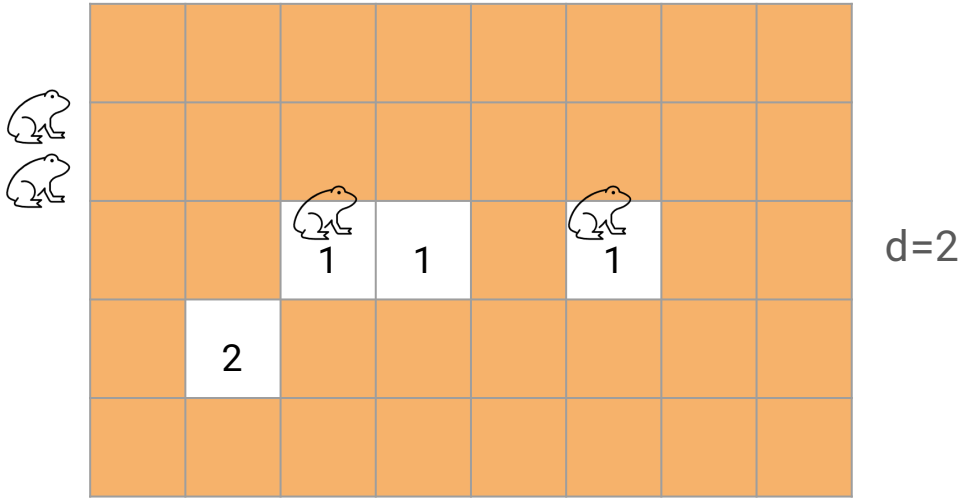
Some frogs are currently in a 2D grid. A frog can jump from its current location to another cell **no more than d units away** (in Euclidean distance). However, leaving a cell causes the cell's height to decrease by 1. A frog cannot jump to a cell with height 0. What is the maximum number of frogs that can jump outside the grid boundaries?



Problem Source: Luogu P2472

Modeling Non-Graph Problems with Flow

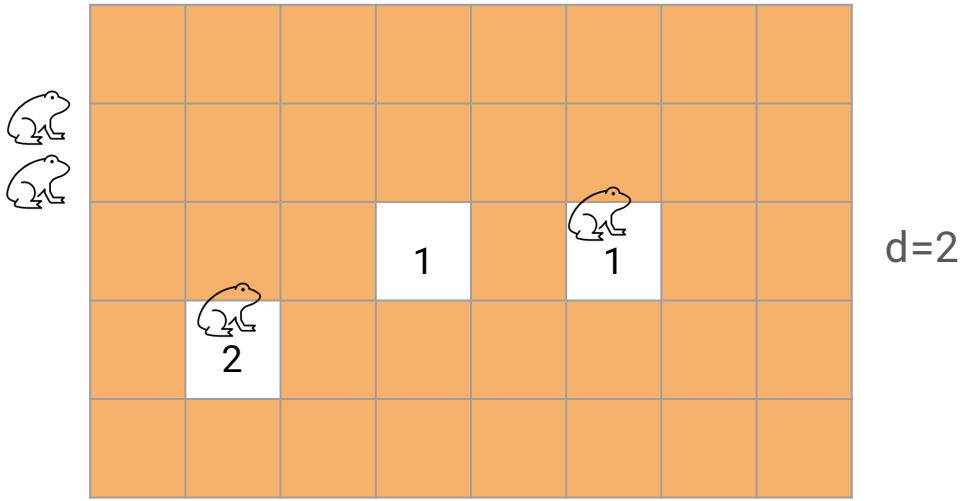
Some frogs are currently in a 2D grid. A frog can jump from its current location to another cell **no more than d units away** (in Euclidean distance). However, leaving a cell causes the cell's height to decrease by 1. A frog cannot jump to a cell with height 0. What is the maximum number of frogs that can jump outside the grid boundaries?



Problem Source: Luogu P2472

Modeling Non-Graph Problems with Flow

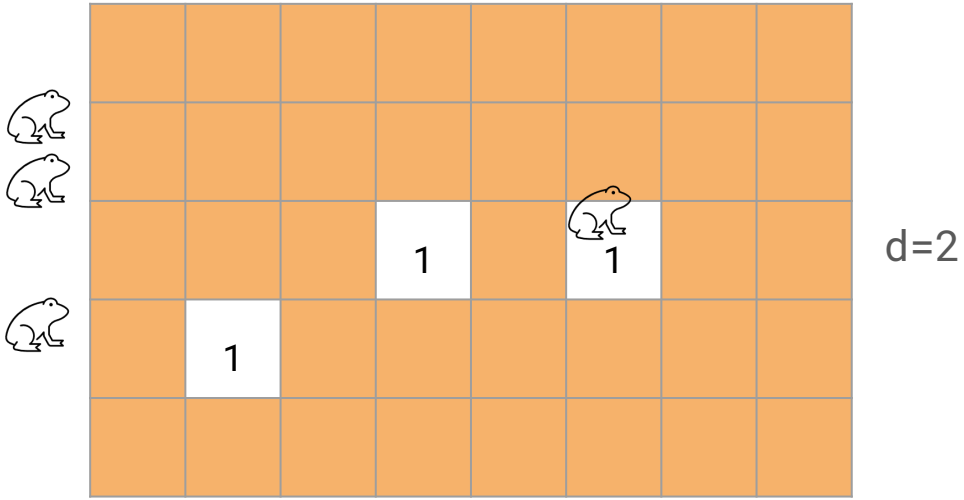
Some frogs are currently in a 2D grid. A frog can jump from its current location to another cell **no more than d units away** (in Euclidean distance). However, leaving a cell causes the cell's height to decrease by 1. A frog cannot jump to a cell with height 0. What is the maximum number of frogs that can jump outside the grid boundaries?



Problem Source: Luogu P2472

Modeling Non-Graph Problems with Flow

Some frogs are currently in a 2D grid. A frog can jump from its current location to another cell **no more than d units away** (in Euclidean distance). However, leaving a cell causes the cell's height to decrease by 1. A frog cannot jump to a cell with height 0. What is the maximum number of frogs that can jump outside the grid boundaries?

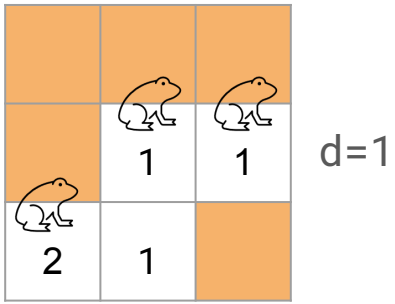


Problem Source: Luogu P2472

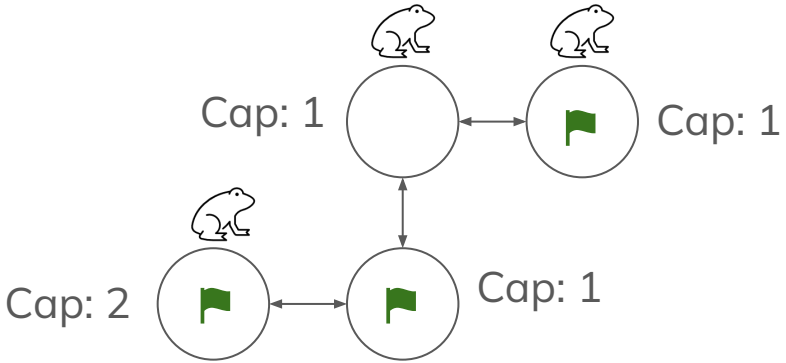
Modeling Non-Graph Problems with Flow

Let's try to model this into a flow problem.

- What does the vertices represent? Grid cells.
- What does the flow represent? Movement of the frogs.



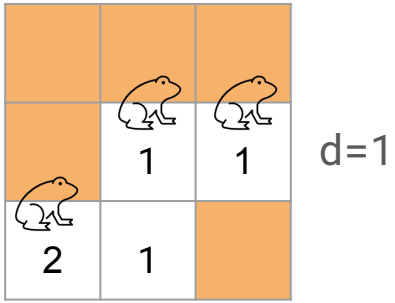
d=1



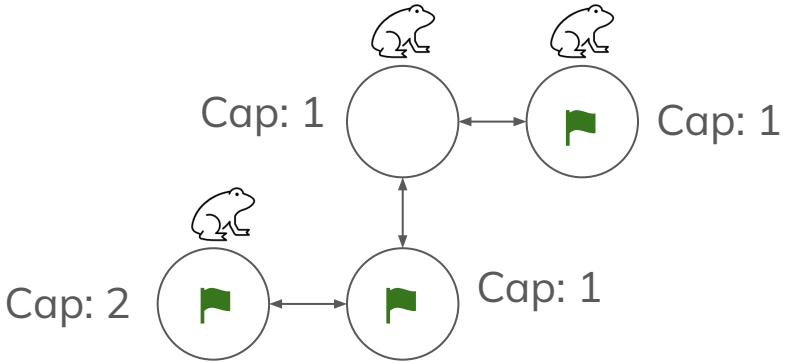
Problem Source: Luogu P2472

Modeling Non-Graph Problems with Flow

Issue 1: There are multiple starting and ending locations.



d=1

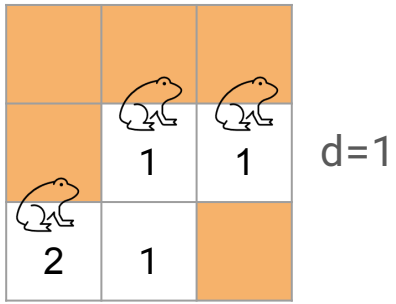


Problem Source: Luogu P2472

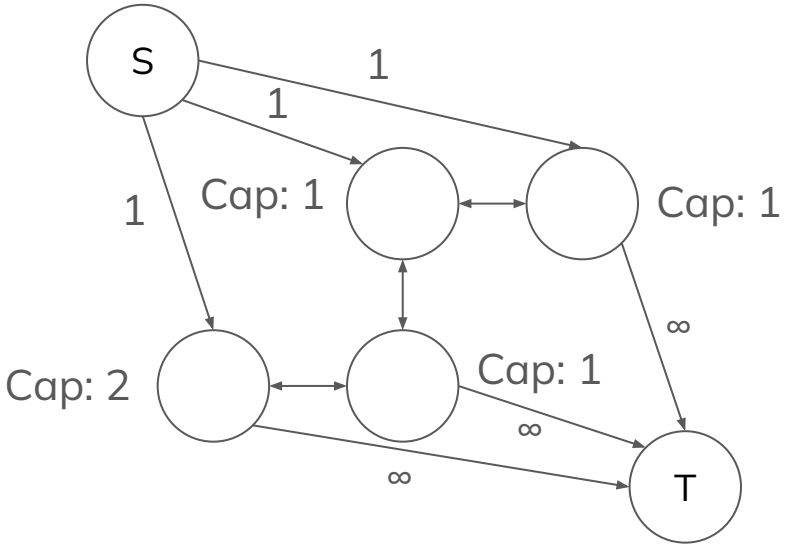
Modeling Non-Graph Problems with Flow

Issue 1: There are multiple starting and ending locations.

- Build a “super source” and a “super sink” connecting those.
- Capacity from source = 1 (the number of frogs). Capacity to sink = ∞.



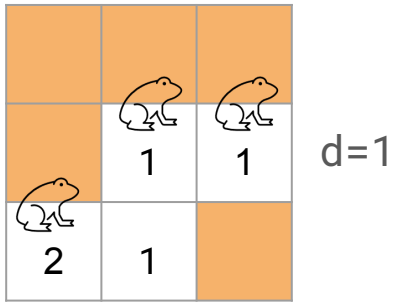
d=1



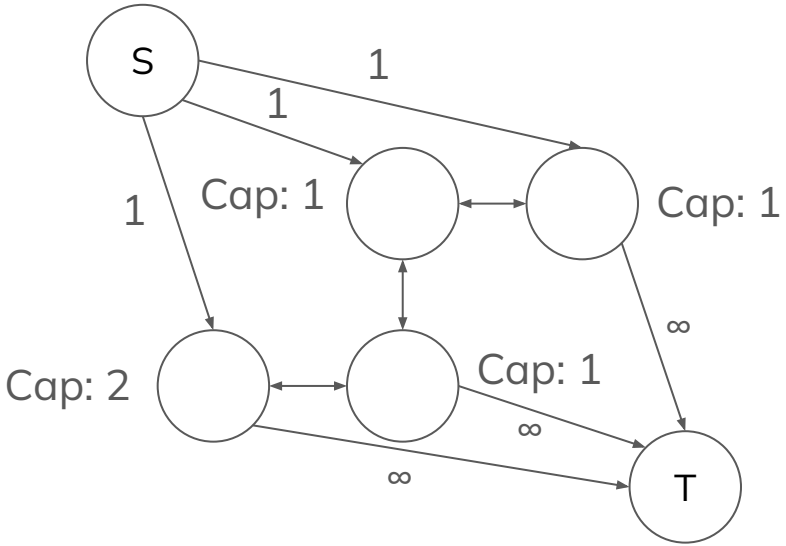
Problem Source: Luogu P2472

Modeling Non-Graph Problems with Flow

Issue 2: There are limits on the number of times each cell is used.



d=1

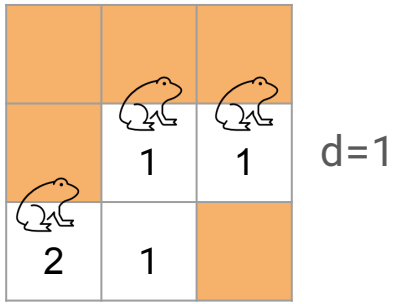


Problem Source: Luogu P2472

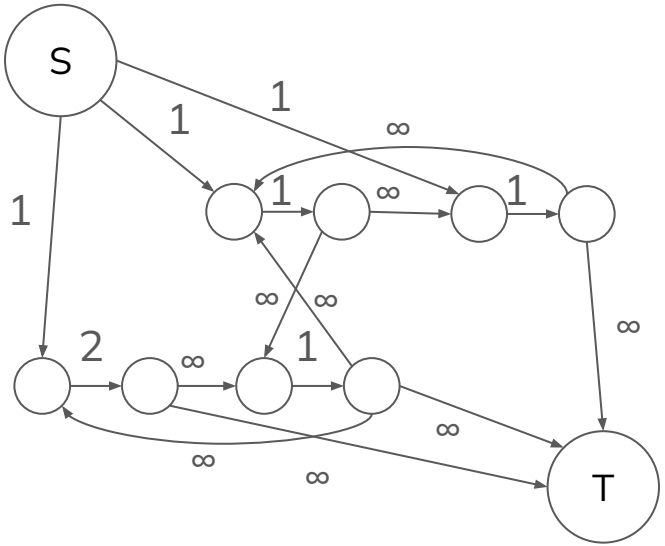
Modeling Non-Graph Problems with Flow

Issue 2: There are limits on the number of times each cell is used.

- Note that these are **vertex capacities.** Use the **vertex splitting technique**.



d=1

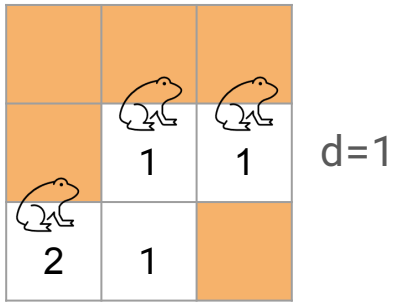


Problem Source: Luogu P2472

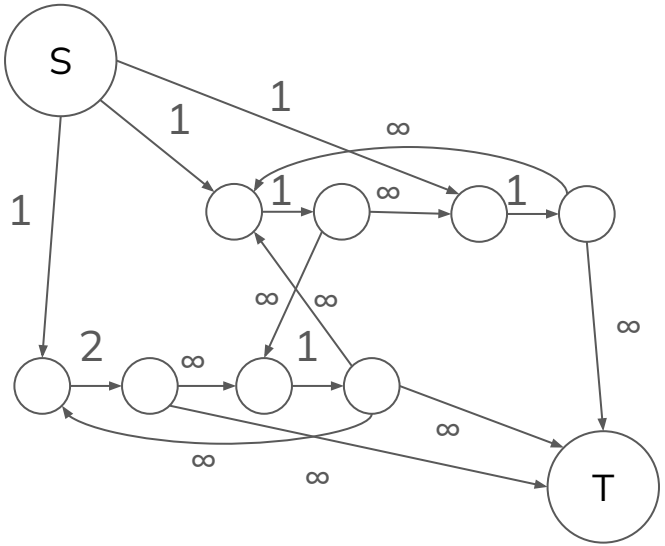
Modeling Non-Graph Problems with Flow

Solution:

- Build the flow network for the given input grid.
- Answer = **Maximum flow** from S to T. // O(EF) = O(Grid Size^2 × Number of frogs)



d=1



Summary

To solve **maximum flow** problems:

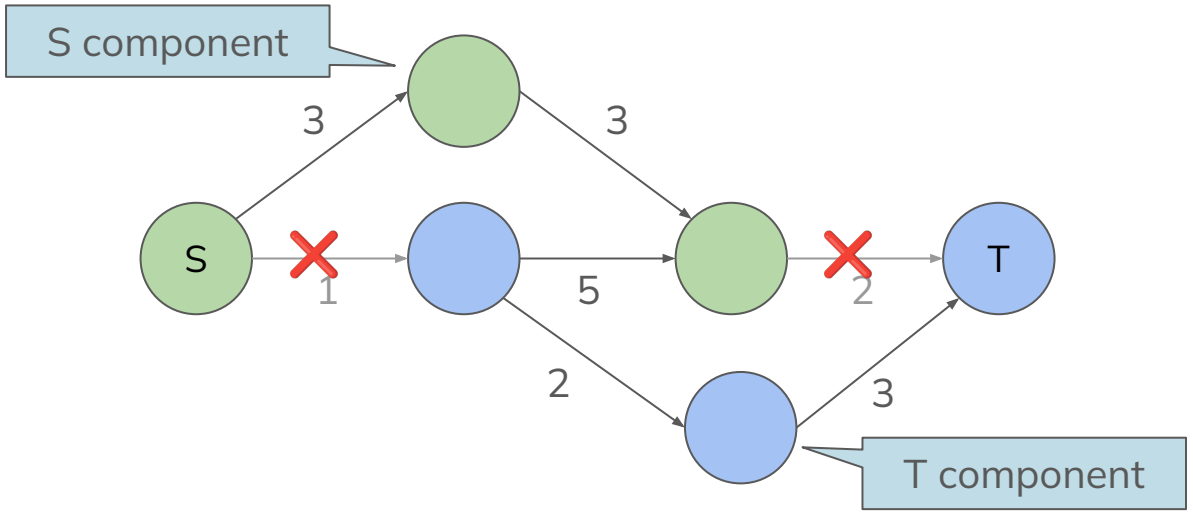
- Find out what the **vertices** and the **flow** (→ edges) should represent.
- Connect all “starting locations” and “ending locations” (goals) with a **super source** and a **super sink**.
- Convert all vertex capacities into edge capacities with the **vertex splitting trick**.
- Run Ford–Fulkerson Algorithm to find the maximum flow.

- HKOJ Practice Tasks: A410, A411, A412

Minimum Cut

Minimum Cut Problem

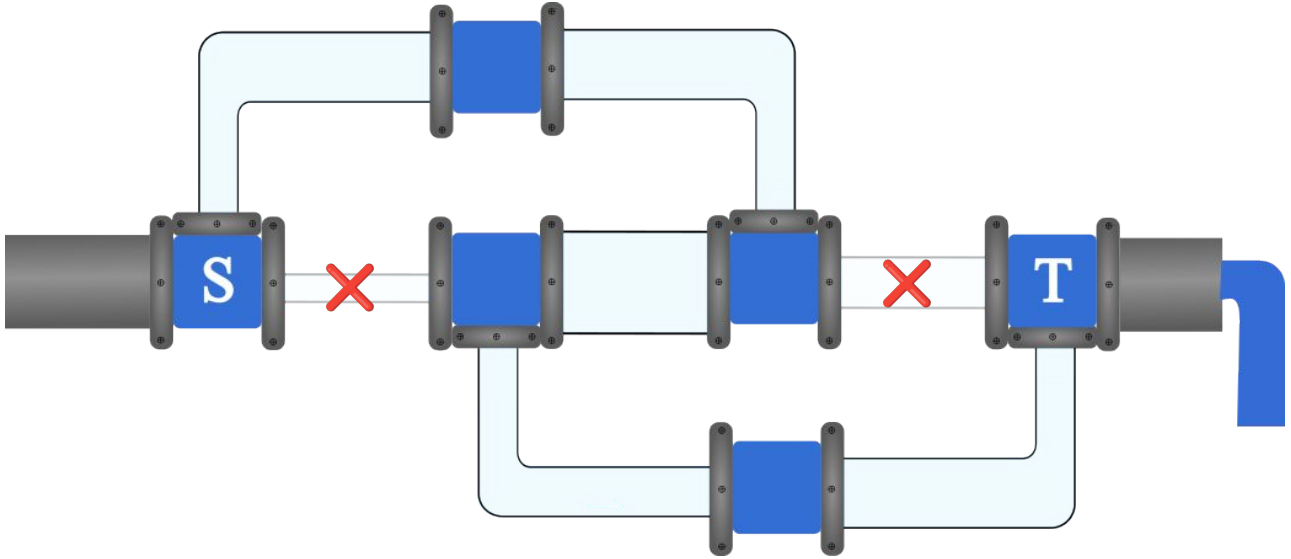
You are given a **weighted** graph. Delete some edges with **smallest total weight** to disconnect the sink from the source (i.e. there are no more paths from S to T).



Max-Flow Min-Cut Theorem

Let's rewrite it using **flow** terms.

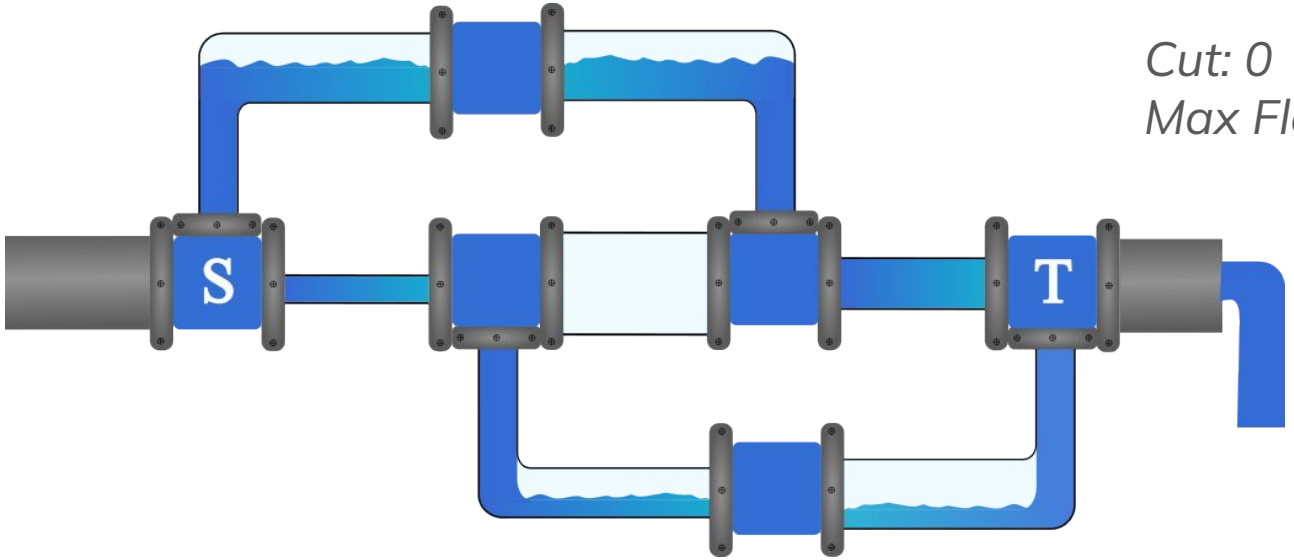
You are given a flow network. Delete some edges with **smallest total capacity** to disconnect the sink from the source (i.e. the maximum flow from S to T is 0).



Max-Flow Min-Cut Theorem

First Impression:

- By cutting an edge with capacity x, we can only reduce the flow by x at best.
- Therefore, **minimum cut ≥ maximum flow**. (They are related!)

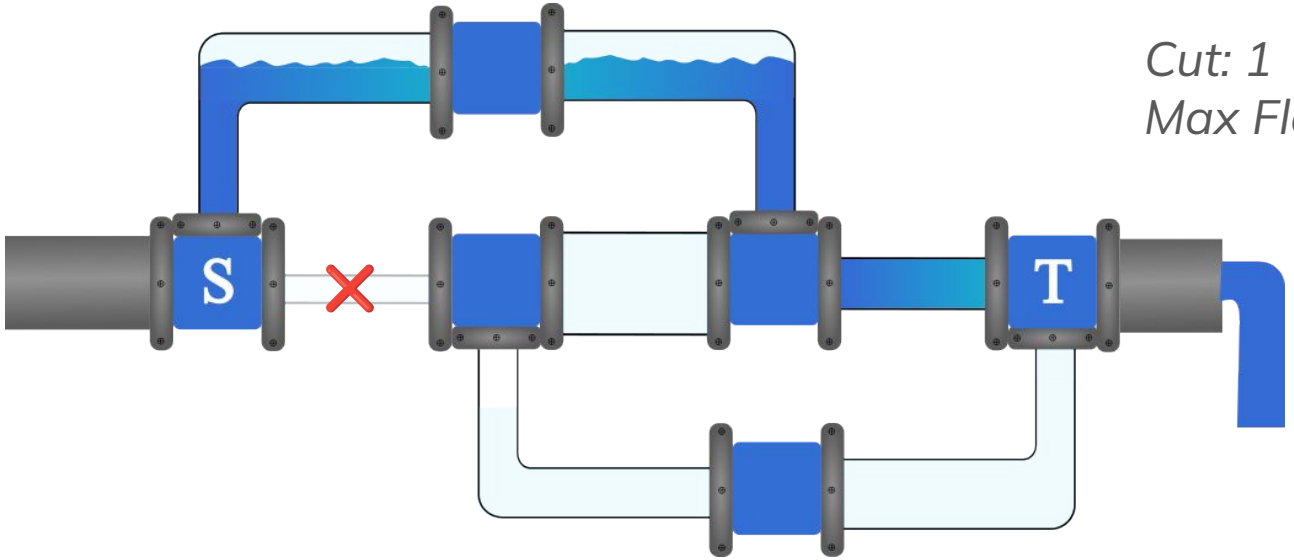


Cut: 0
Max Flow: 3

Max-Flow Min-Cut Theorem

First Impression:

- By cutting an edge with capacity x, we can only reduce the flow by x at best.
- Therefore, **minimum cut ≥ maximum flow**. (They are related!)

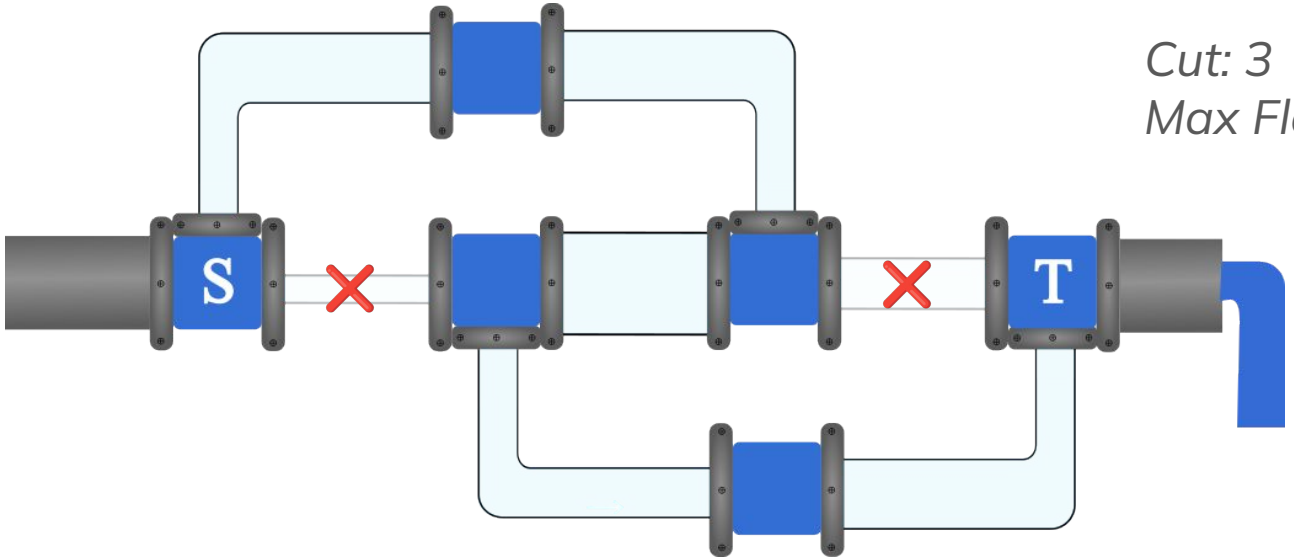


Cut: 1
Max Flow: 2

Max-Flow Min-Cut Theorem

First Impression:

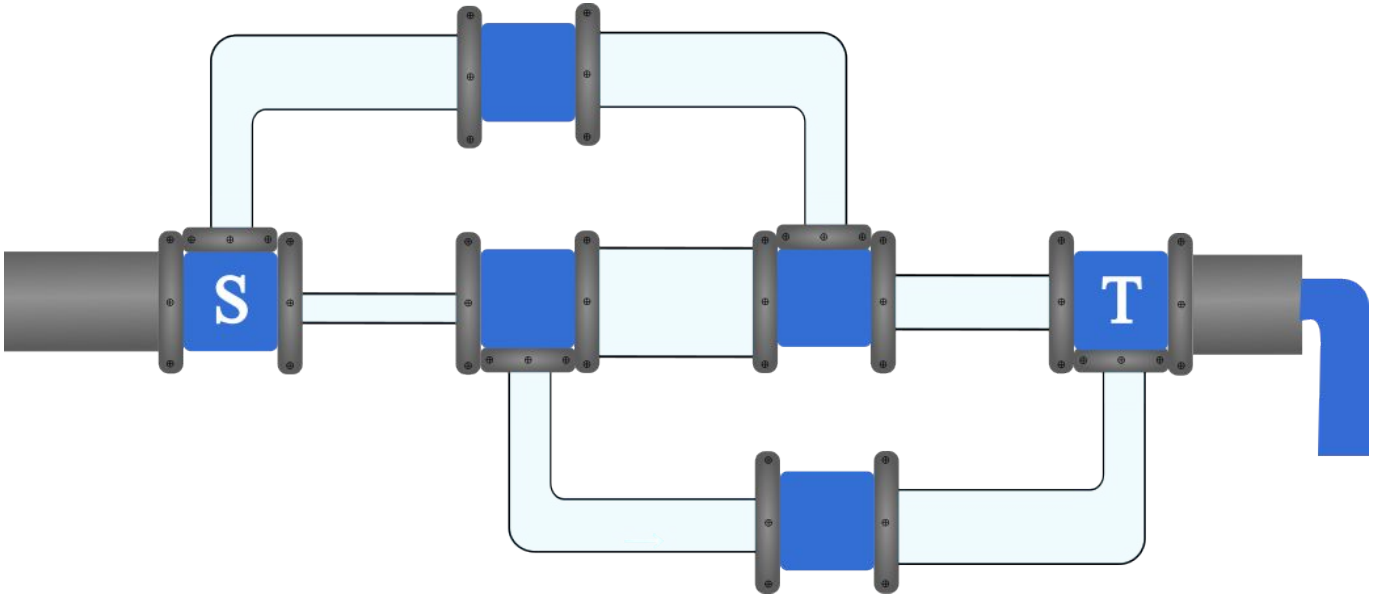
- By cutting an edge with capacity x, we can only reduce the flow by x at best.
- Therefore, **minimum cut ≥ maximum flow**. (They are related!)



Cut: 3
Max Flow: 0

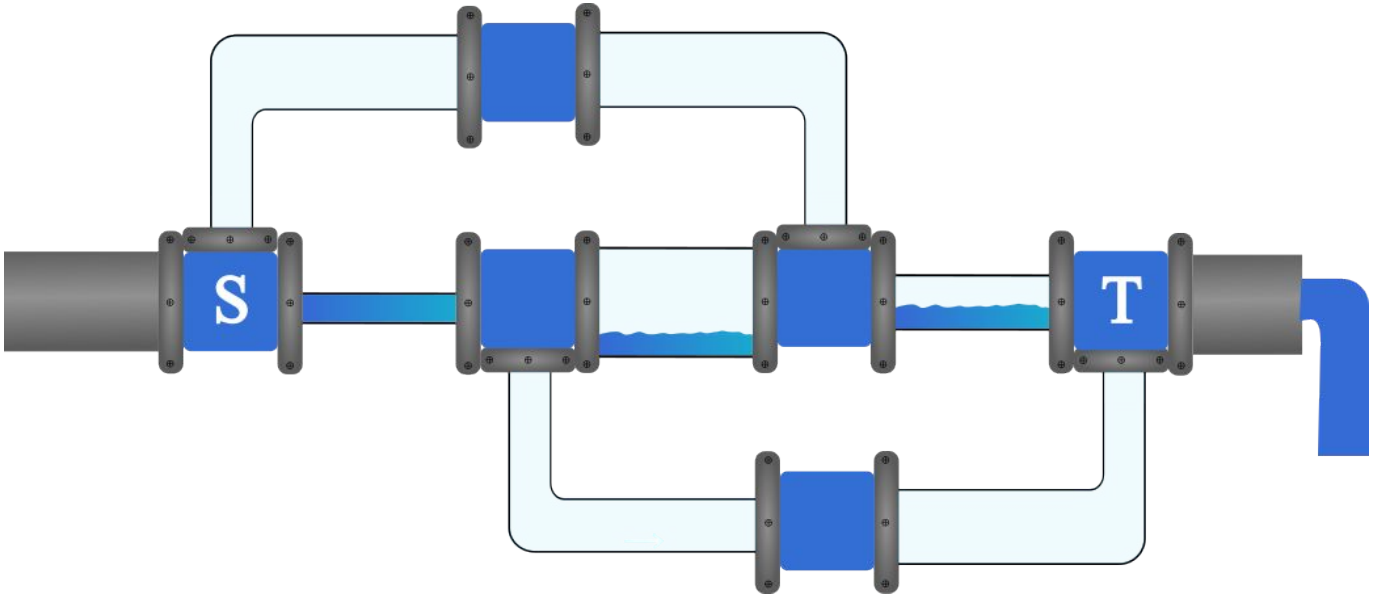
Max-Flow Min-Cut Theorem

Can we find the **minimum cut** from the **maximum flow**?



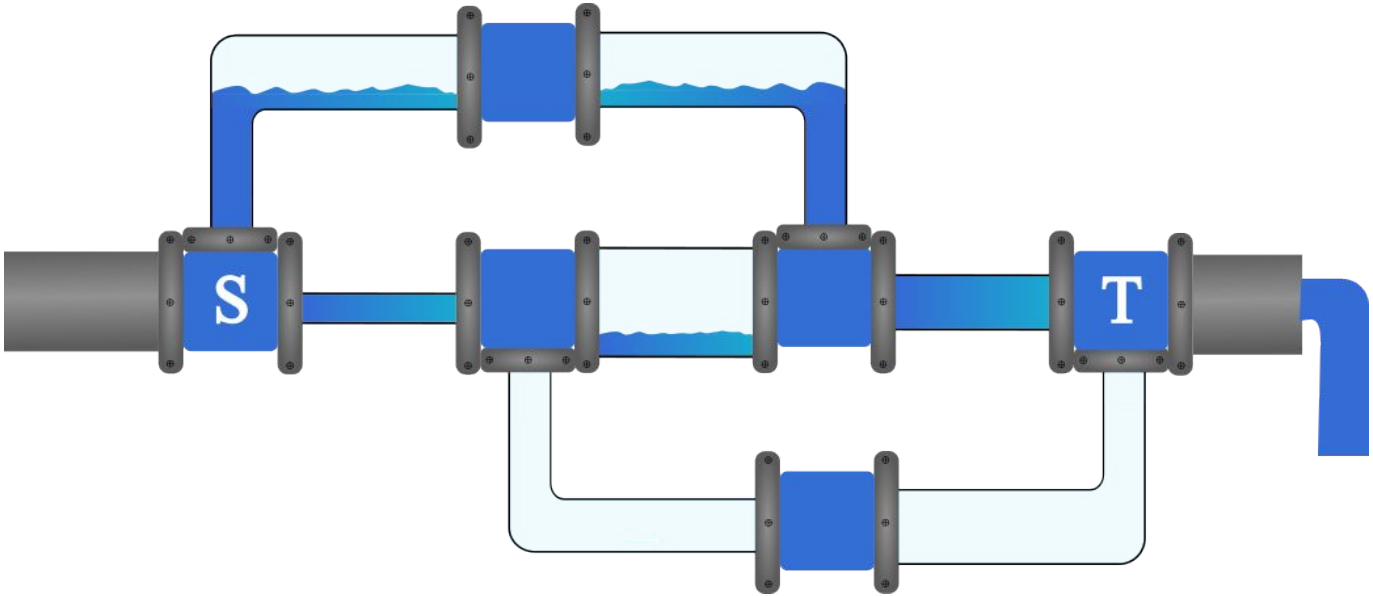
Max-Flow Min-Cut Theorem

Can we find the **minimum cut** from the **maximum flow**?



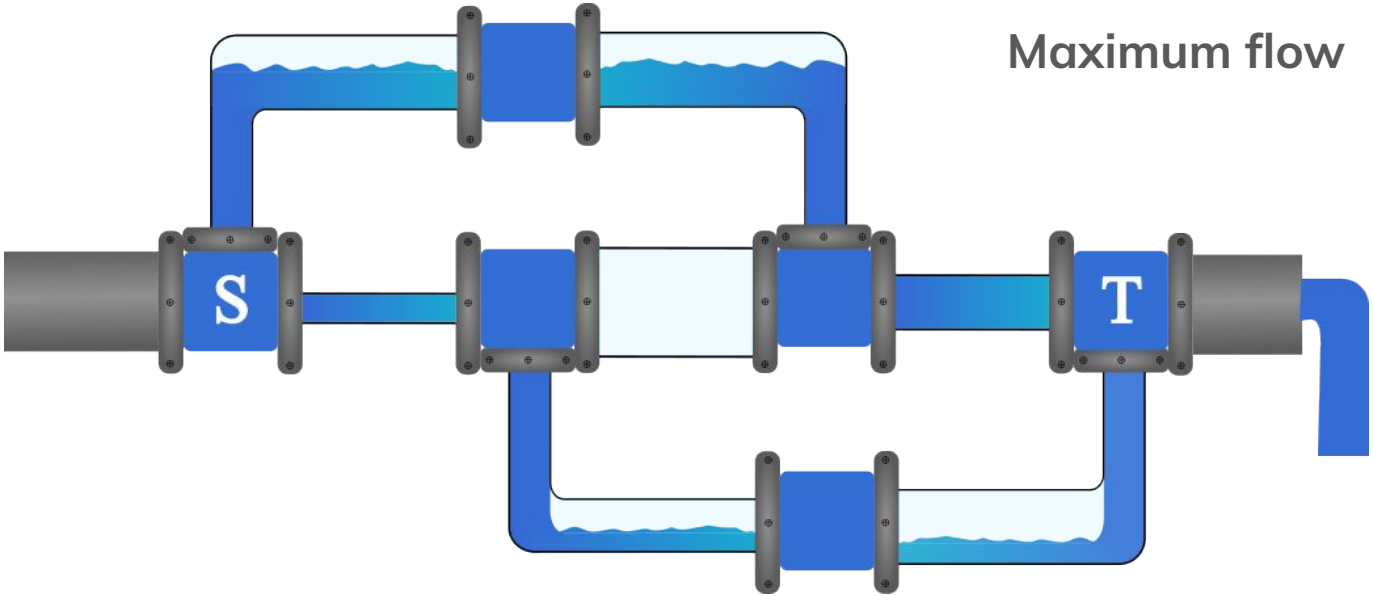
Max-Flow Min-Cut Theorem

Can we find the **minimum cut** from the **maximum flow**?



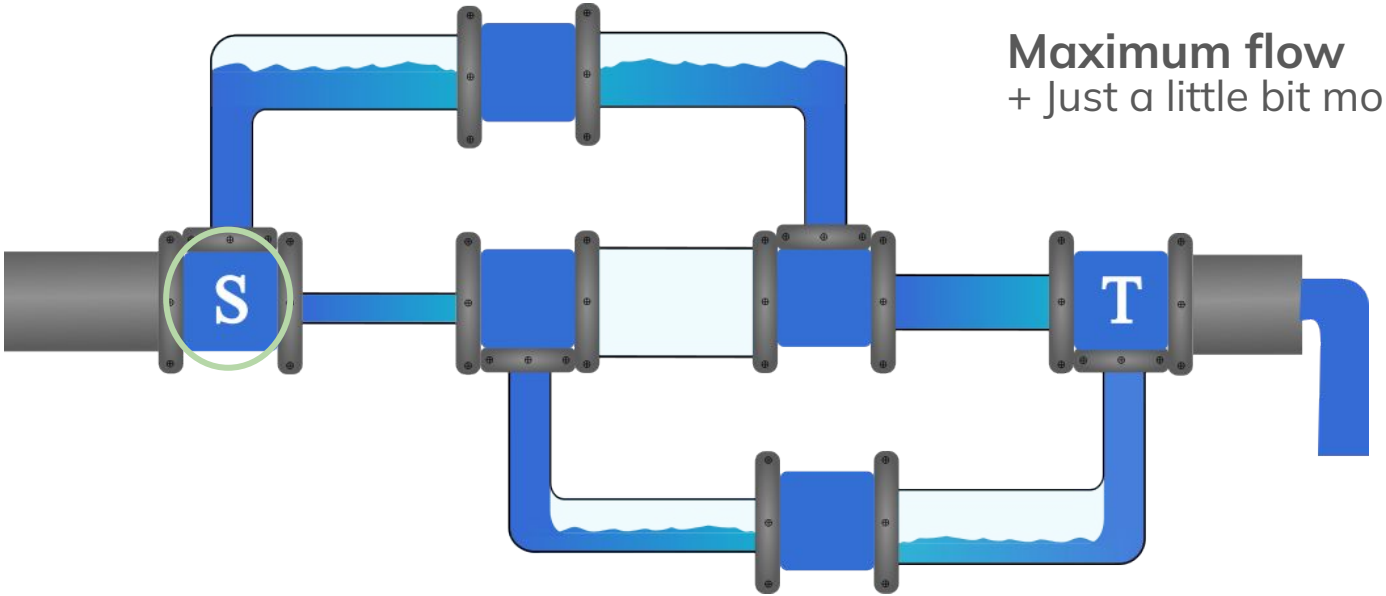
Max-Flow Min-Cut Theorem

Can we find the **minimum cut** from the **maximum flow**?



Max-Flow Min-Cut Theorem

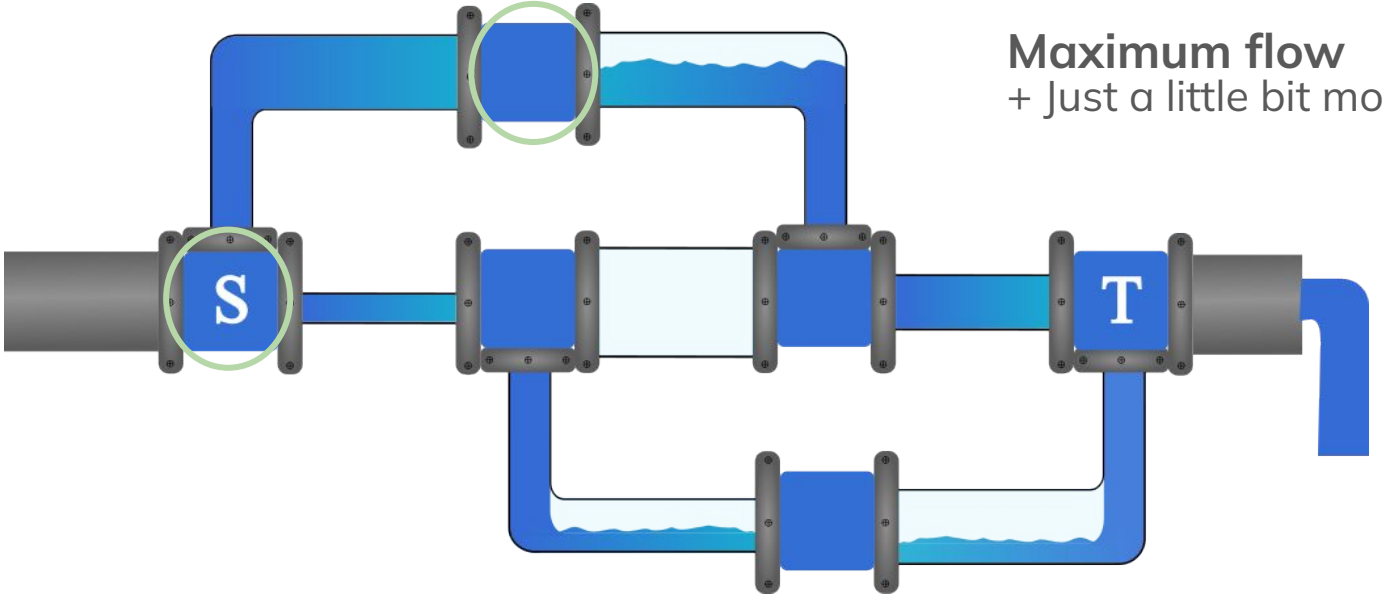
Can we find the **minimum cut** from the **maximum flow**?



Maximum flow
+ Just a little bit more...

Max-Flow Min-Cut Theorem

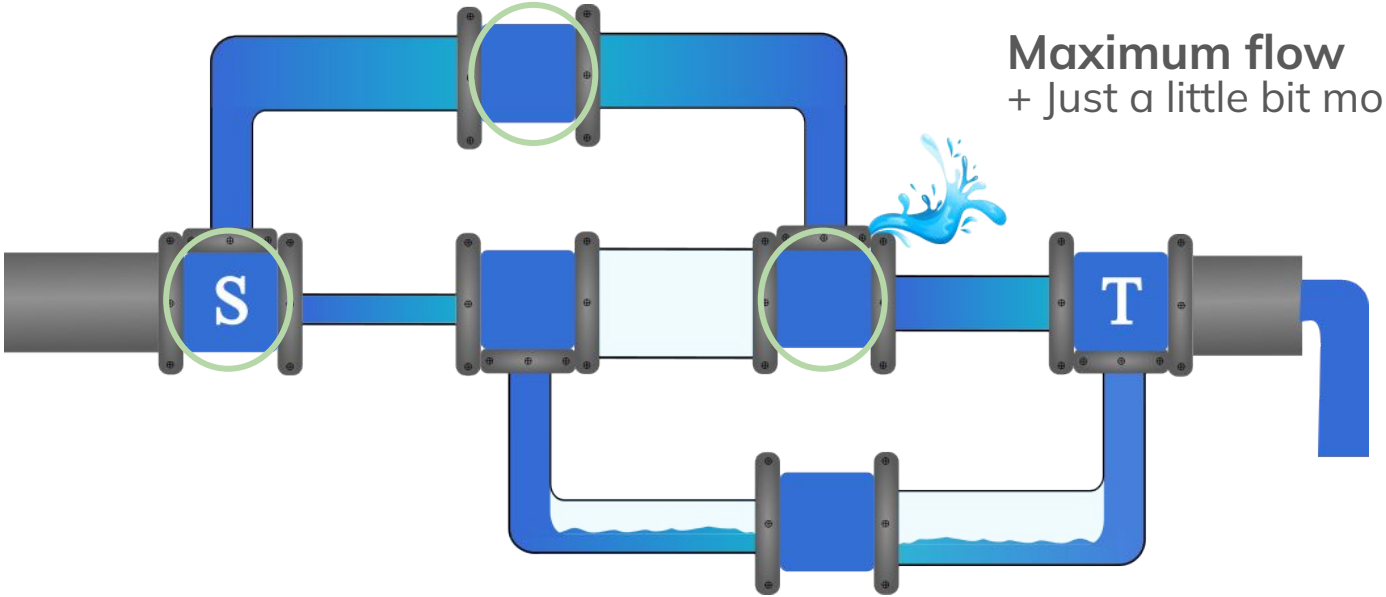
Can we find the **minimum cut** from the **maximum flow**?



Maximum flow
+ Just a little bit more...

Max-Flow Min-Cut Theorem

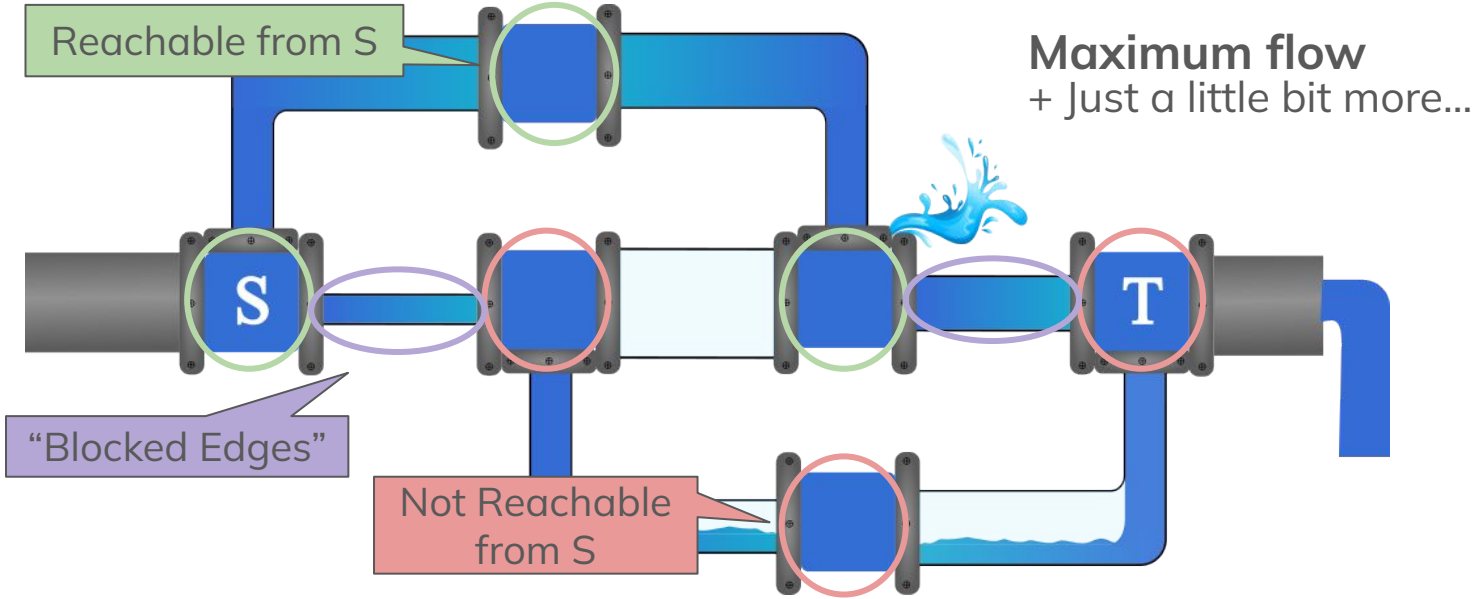
Can we find the **minimum cut** from the **maximum flow**?



Maximum flow
+ Just a little bit more...

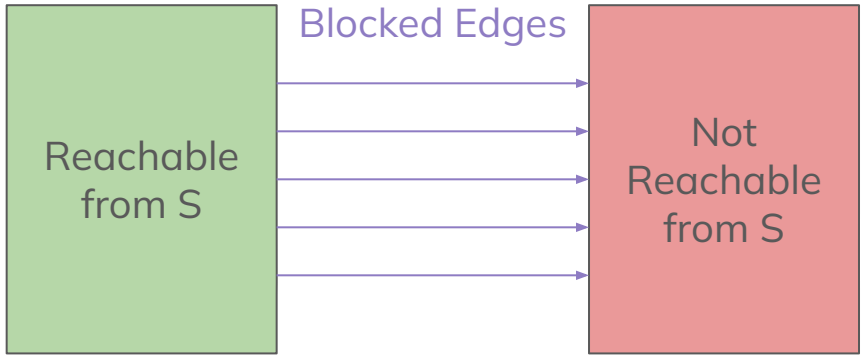
Max-Flow Min-Cut Theorem

Can we find the **minimum cut** from the **maximum flow**?



Max-Flow Min-Cut Theorem

Can we find the **minimum cut** from the **maximum flow**?

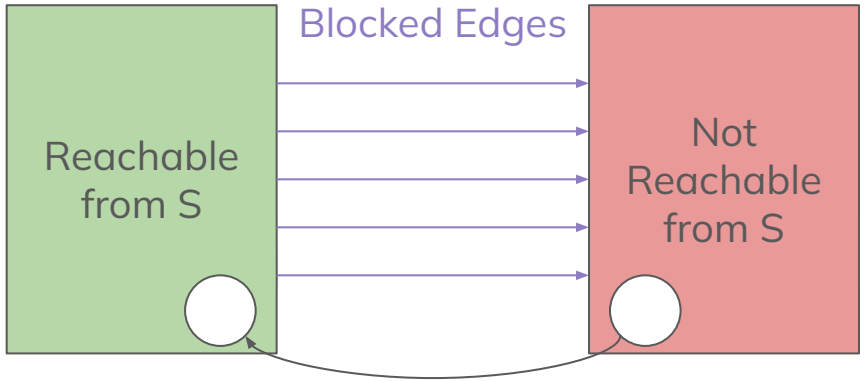


Observation 1: The flow in all blocked edges are at their maximum capacities.

(Otherwise, they do not “block”.)

Max-Flow Min-Cut Theorem

Can we find the **minimum cut** from the **maximum flow**?



Can we have flow in this direction?

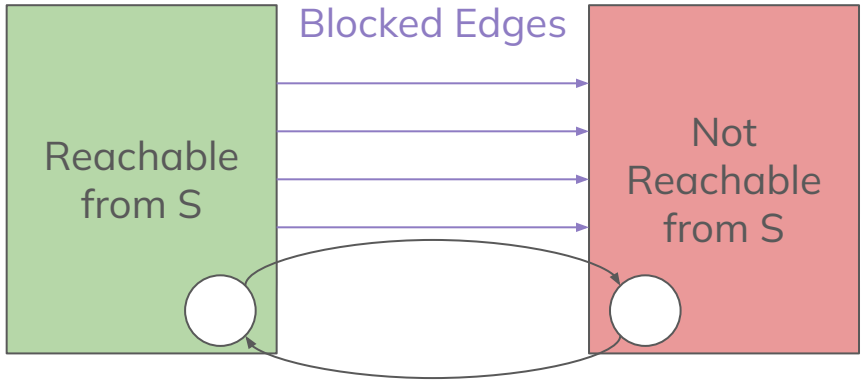
Observation 1: The flow in all blocked edges are at their maximum capacities.

(Otherwise, they do not "block".)

Observation 2: If there are flow from red to green area, we can "undo" it to "expand" the green area accordingly.

Max-Flow Min-Cut Theorem

Can we find the **minimum cut** from the **maximum flow**?



Can we have flow in this direction?

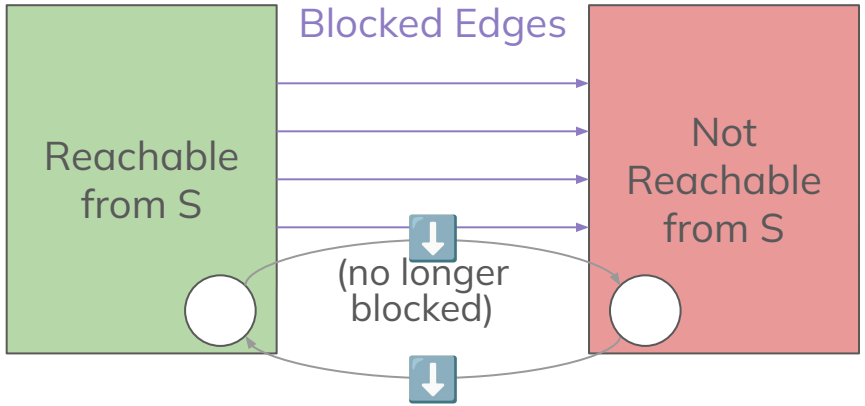
Observation 1: The flow in all blocked edges are at their maximum capacities.

(Otherwise, they do not "block".)

Observation 2: If there are flow from red to green area, we can "undo" it to "expand" the green area accordingly.

Max-Flow Min-Cut Theorem

Can we find the **minimum cut** from the **maximum flow**?



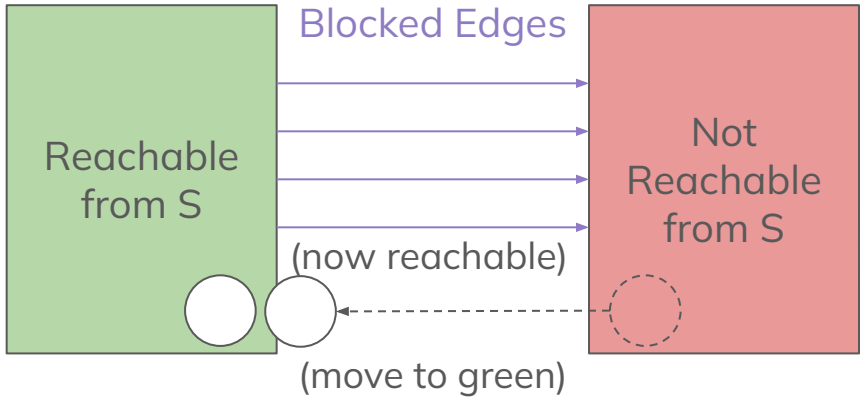
Observation 1: The flow in all blocked edges are at their maximum capacities.

(Otherwise, they do not "block".)

Observation 2: If there are flow from red to green area, we can "undo" it to "expand" the green area accordingly.

Max-Flow Min-Cut Theorem

Can we find the **minimum cut** from the **maximum flow**?



Can we have flow in this direction?

Observation 1: The flow in all blocked edges are at their maximum capacities.

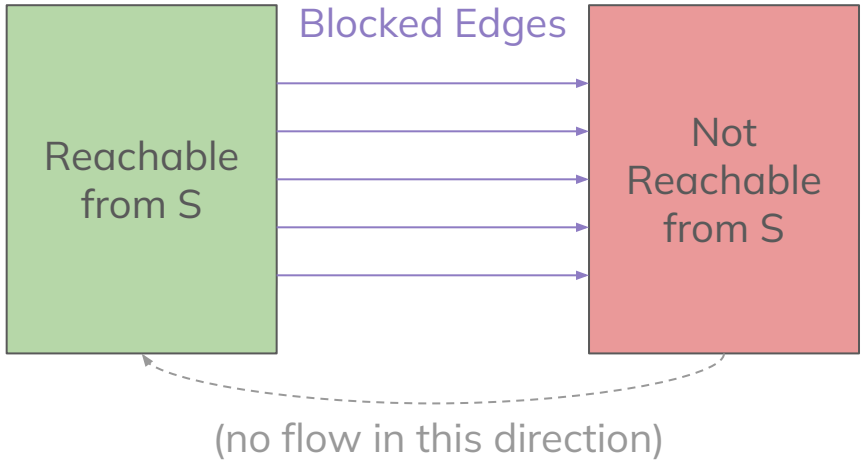
(Otherwise, they do not "block".)

Observation 2: If there are flow from red to green area, we can "undo" it to "expand" the green area accordingly.

(...we can repeat this process until no more flow from red to green)

Max-Flow Min-Cut Theorem

Can we find the **minimum cut** from the **maximum flow**?



Observation 1: The flow in all blocked edges are at their maximum capacities.

(Otherwise, they do not "block".)

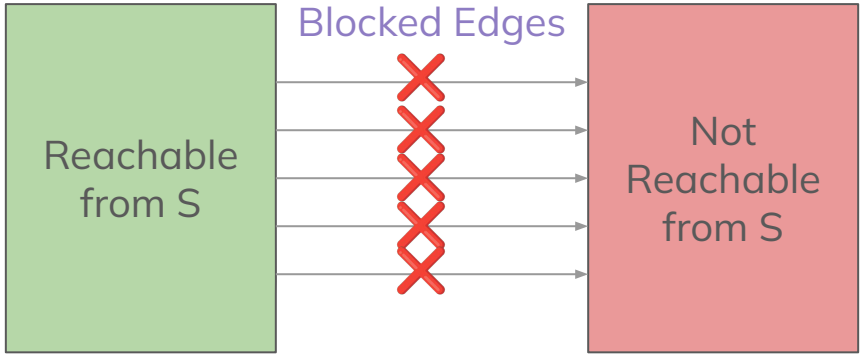
Observation 2: If there are flow from red to green area, we can "undo" it to "expand" the green area accordingly.

Result: The sum of weights of all blocked edges = The maximum flow.

(By flow conservation, flow does not magically "appear" or "disappear" in the red area.)

Max-Flow Min-Cut Theorem

- By cutting all blocked edges, we can attain a cut equal to the maximum flow.
- Is this optimal?
  Yes, since minimum cut ≥ maximum flow.



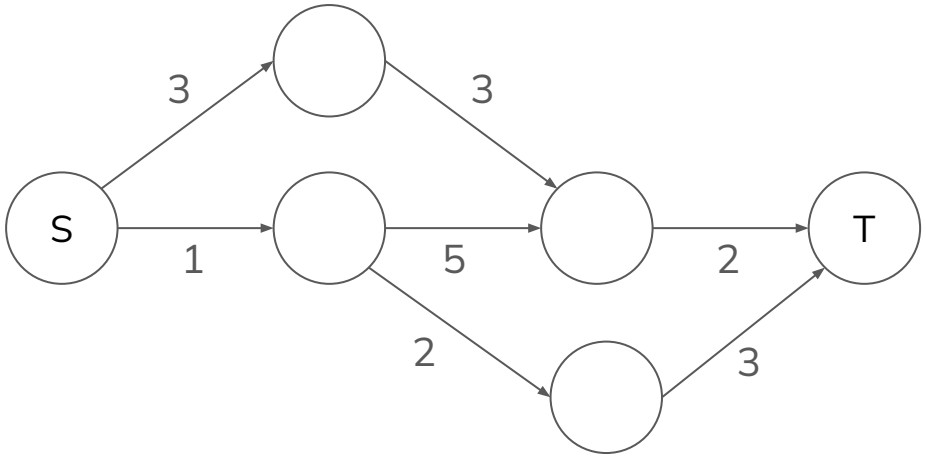
Max-Flow Min-Cut Theorem

The maximum flow from S to T is **equal to** the minimum cut from S to T.

Minimum Cut – Implementation

1. Find the maximum flow (Minimum cut = Maximum flow)
2. Find all vertices reachable / not reachable from S in the **residual network**
3. The cut consists of all edges connecting from a vertex reachable from S to a vertex not reachable from S

[Original Network]



Minimum Cut – Implementation

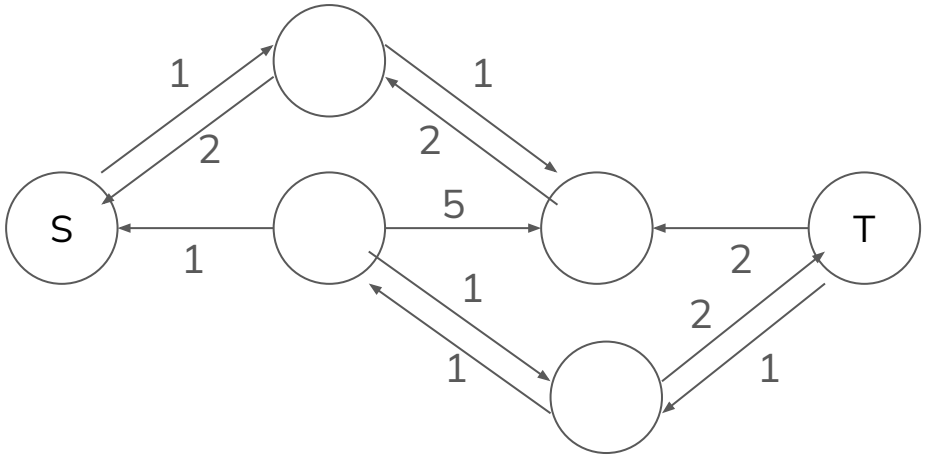
1. Find the maximum flow
   (Minimum cut = Maximum flow)

2. Find all vertices reachable / not reachable from S in the **residual network**

3. The cut consists of all edges connecting from a vertex reachable from S to a vertex not reachable from S

[Residual Network]

Max Flow = 3

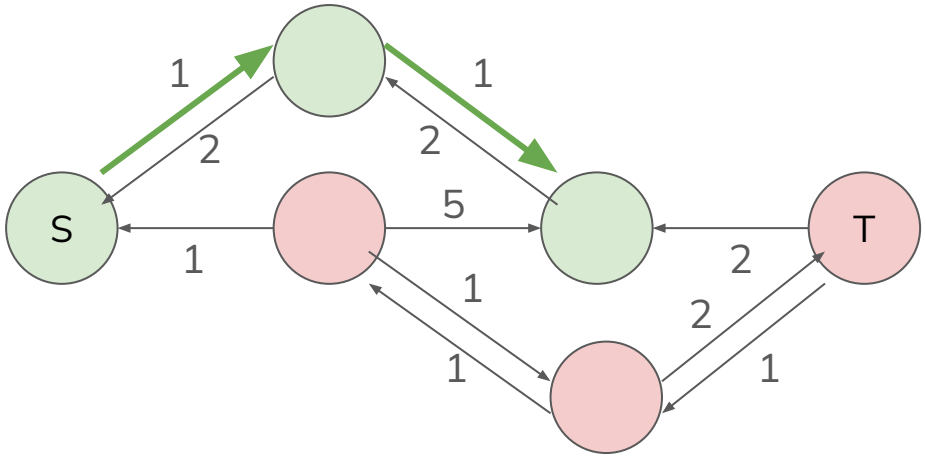


Minimum Cut – Implementation

1. Find the maximum flow (Minimum cut = Maximum flow)
2. Find all vertices reachable / not reachable from S in the **residual network**
3. The cut consists of all edges connecting from a vertex reachable from S to a vertex not reachable from S

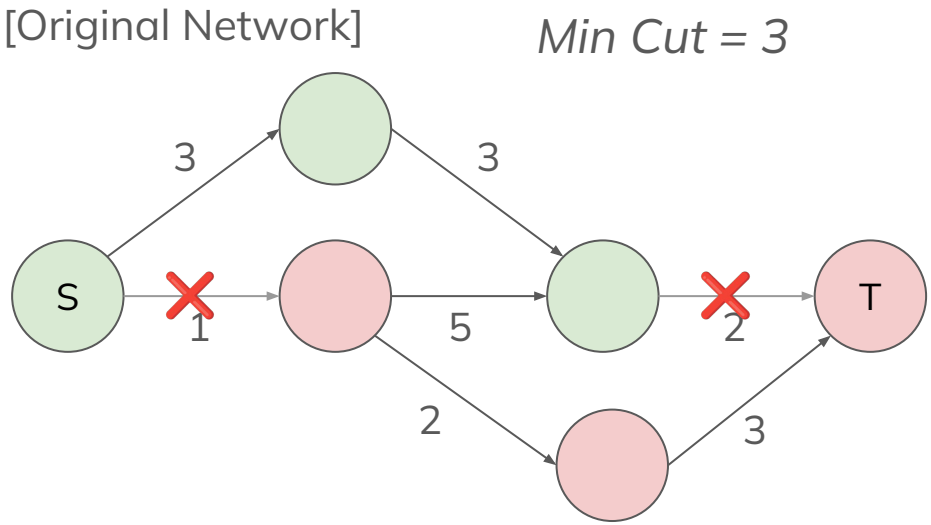
[Residual Network]

Max Flow = 3



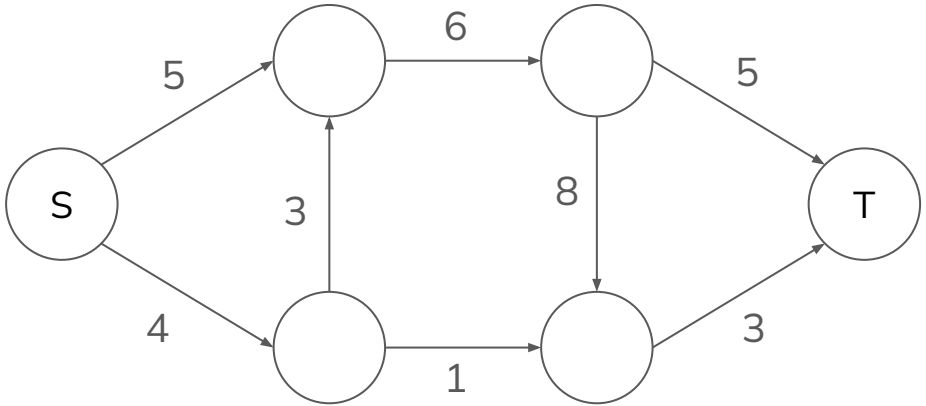
Minimum Cut – Implementation

1. Find the maximum flow (Minimum cut = Maximum flow)
2. Find all vertices reachable / not reachable from S in the **residual network**
3. The cut consists of all edges connecting from a vertex reachable from S to a vertex not reachable from S



Quick Check

🔤 Consider the same flow network. Which edges are in the minimum cut from S to T?



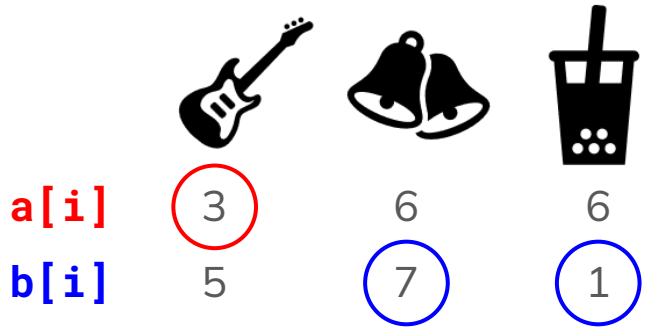
(Answer: 1 and 6)

Set Partitioning Model

There are N items. You should place each item into either the **red** bag or the **blue** bag.

- Putting item `i` in the red bag costs `a[i]`.
- Putting item `i` in the blue bag costs `b[i]`.
- There are `M` constraint pairs `(u, v, w)`. If item `u` is red and item `v` is blue, there is an additional cost of `w`.

What is the minimum possible cost to place all N items?



Constraint Pairs:

1.  `w = 1`
2.  `w = 10`

Set Partitioning Model

To model this as a minimum cut problem, we need to think in a special way:

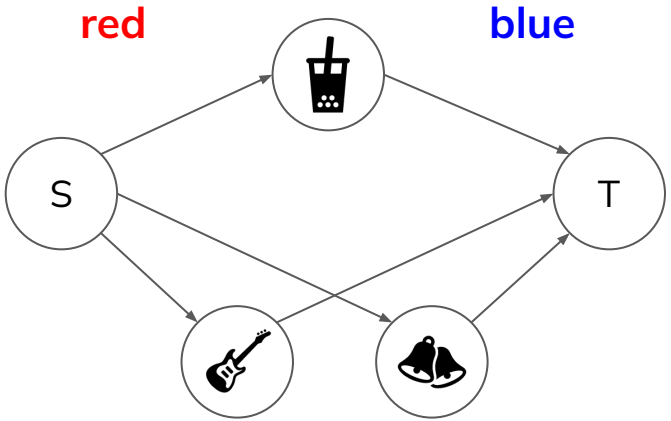
"There will be a path from S to T **if something goes wrong**."

- **Cutting an edge** is same as **paying some cost**.
- Our goal is to disconnect T from S → nothing goes wrong.

Set Partitioning Model

Our Requirements:

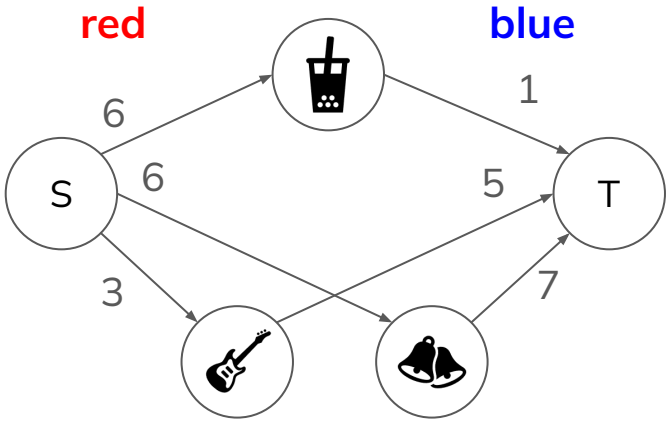
1. Each item must be placed in either the red bag or the blue bag.
   Create a node for each item. Connect it to both S and T.
   Cutting an edge → placing the item into a bag.



Set Partitioning Model

Our Requirements:

1. Each item must be placed in either the red bag or the blue bag.
   Create a node for each item. Connect it to both S and T.
   Cutting an edge → placing the item into a bag.

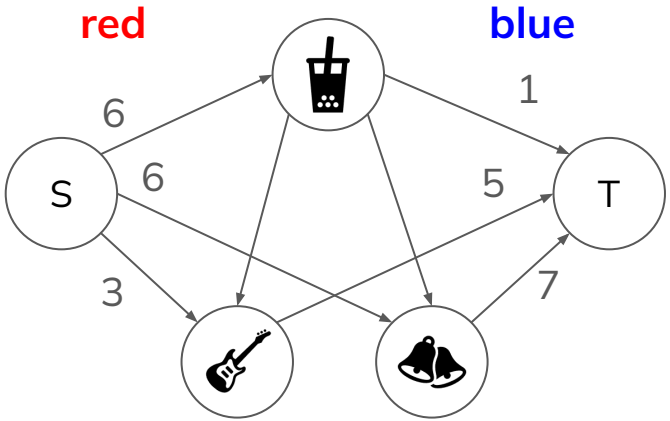


Cost of cutting the edge
= Cost of placing the item

Set Partitioning Model

Our Requirements:

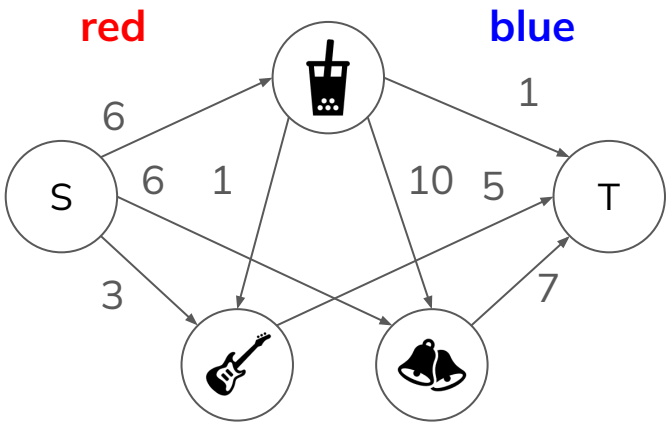
2. There are no conflicts **OR** we paid for the conflicts.
   If item u is **red** and item v is **blue**, then we have the edges S → u and v → T **cut** and the edges u → T and S → v **remaining**. Build an edge v → u to connect from S to T.



Set Partitioning Model

Our Requirements:

2. There are no conflicts **OR** we paid for the conflicts.
   If item u is **red** and item v is **blue**, then we have the edges S → u and v → T **cut** and the edges u → T and S → v **remaining**. Build an edge v → u to connect from S to T.

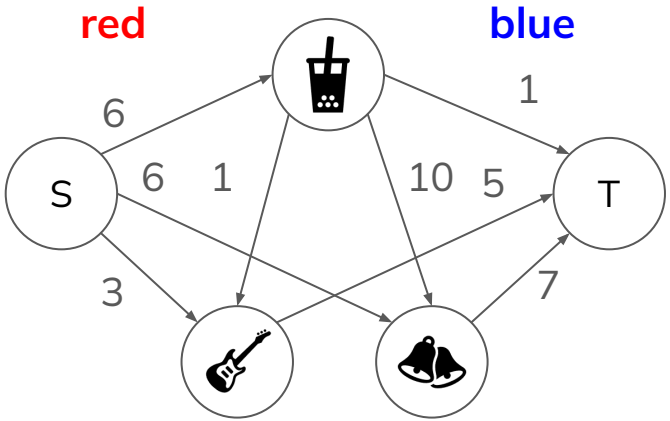


Cost of cutting the edge
= Cost of the conflict

Set Partitioning Model

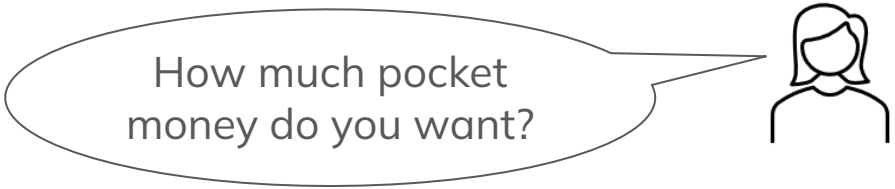
Solution:

- For each item, build an edge S → `i` with weight `a[i]` and `i` → T with weight `b[i]`.
- For each constraint, build an edge v → u with weight `w`.
- Answer = Minimum Cut = Maximum Flow.



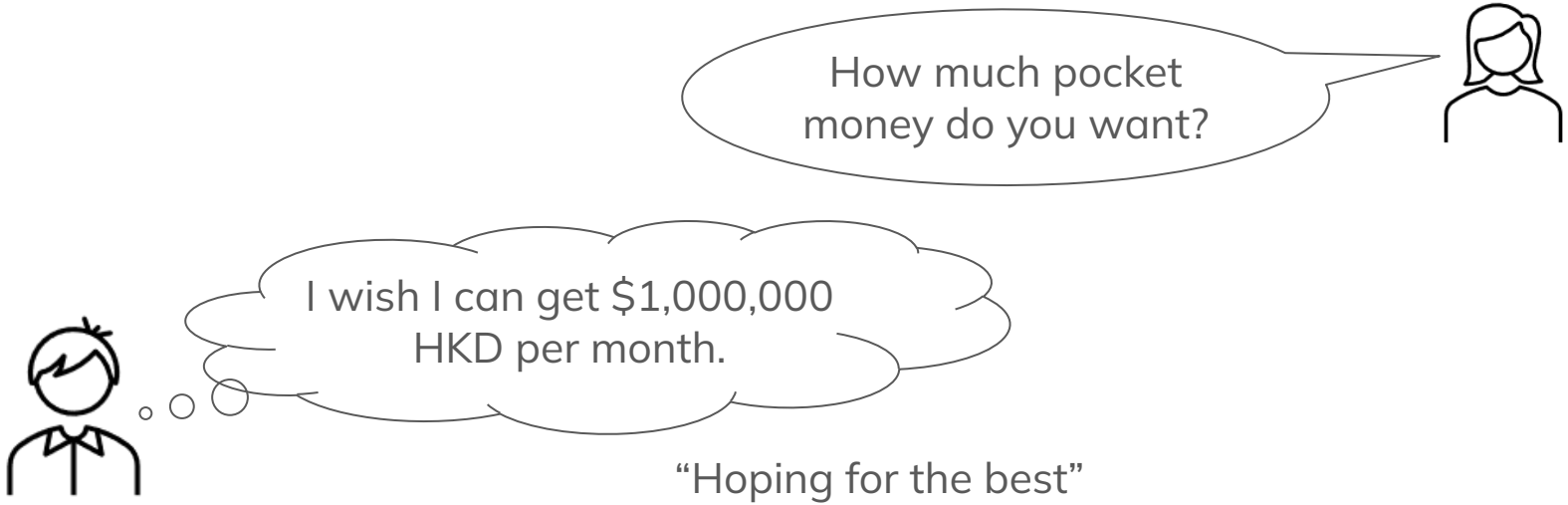
“Optimistic Thinking”

Imagine you are negotiating for pocket money with your parents.



"Optimistic Thinking"

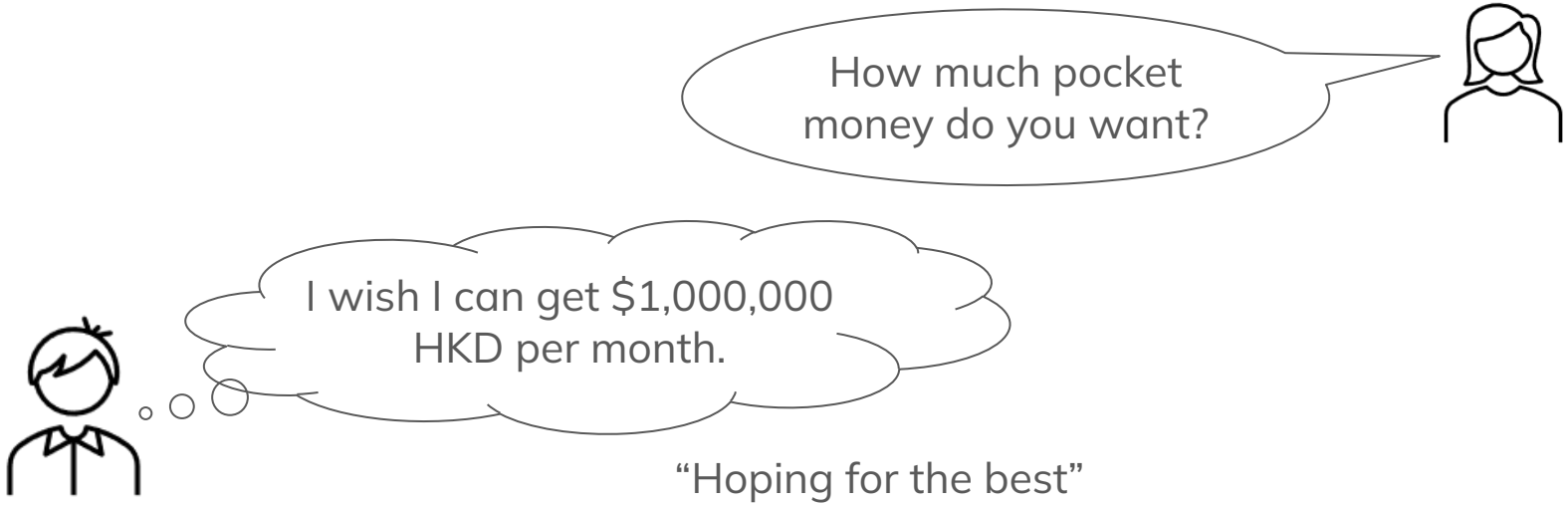
Imagine you are negotiating for pocket money with your parents.



"Hoping for the best"

"Optimistic Thinking"

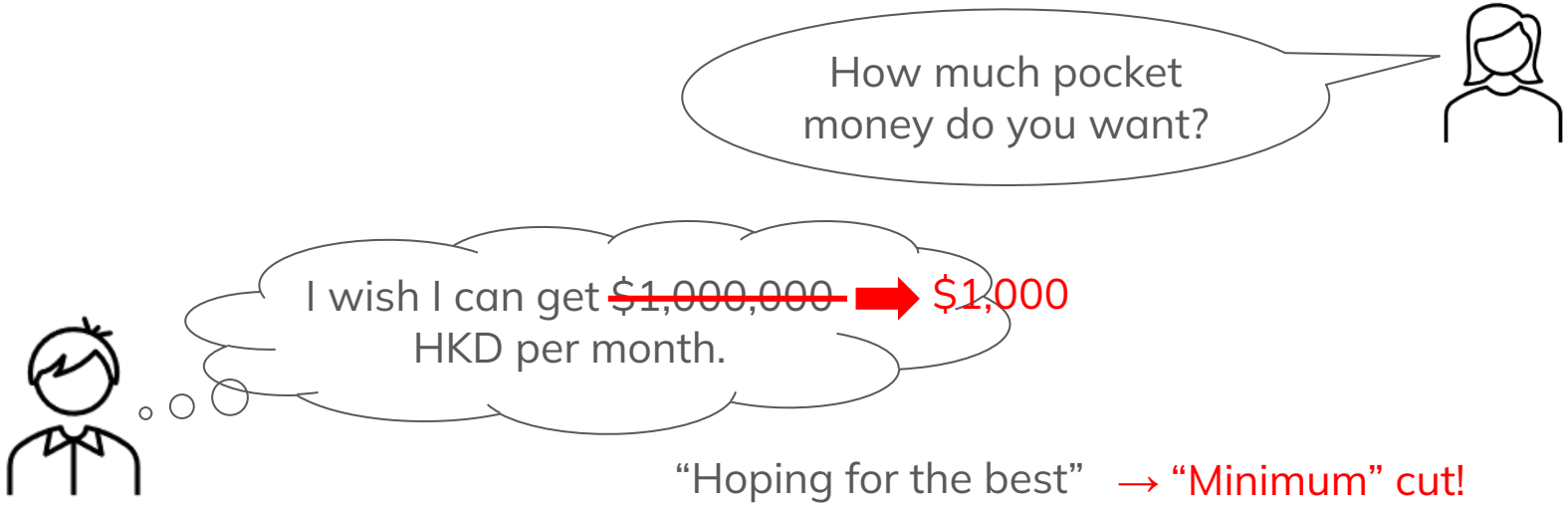
Imagine you are negotiating for pocket money with your parents.



"Hoping for the best"

"Optimistic Thinking"

Imagine you are negotiating for pocket money with your parents.



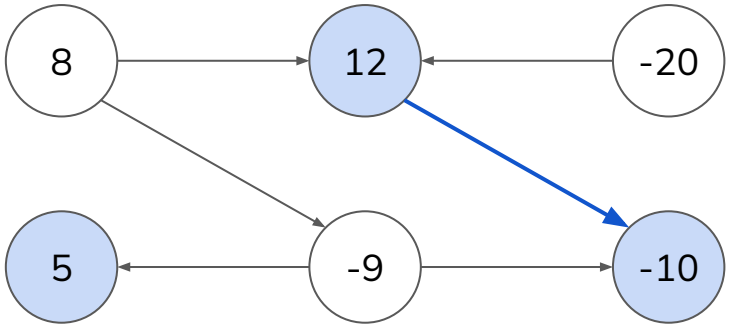
"Hoping for the best" → "Minimum" cut!

Modeling Problems with Cut

Maximum Weight Closure: Given a **directed** graph where each **vertex** has a **weight**, pick some vertices subject to the following condition for each edge u → v:

"If you picked vertex u, you must pick vertex v as well."

Maximize the sum of weights of the vertices picked (the weights can be negative).

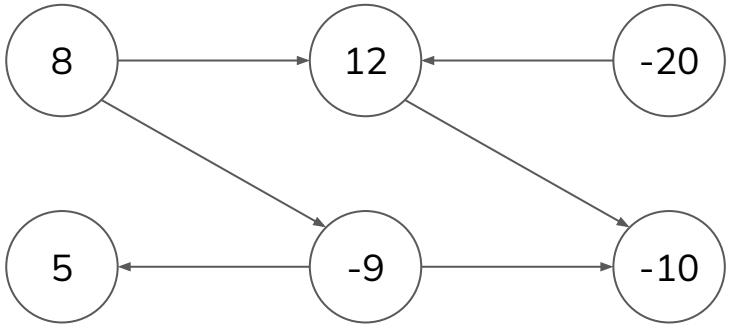


Sum = 12 + 5 - 10 = **7**

Modeling Problems with Cut

Let's apply "optimistic thinking":

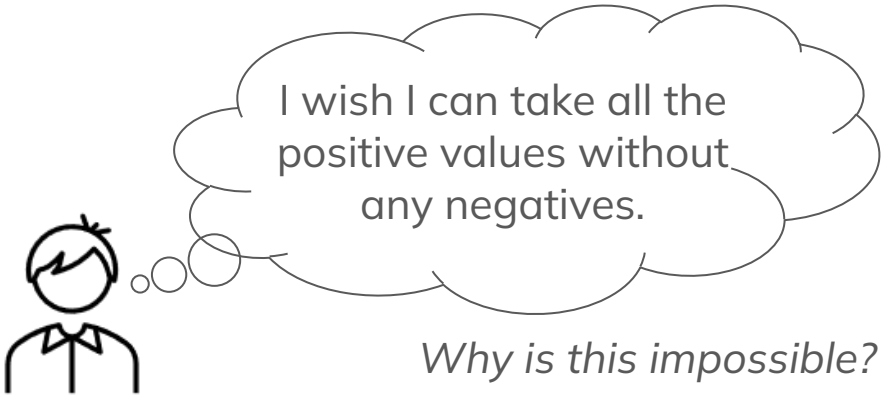
What is the ***best ever scenario*** you can wish for?



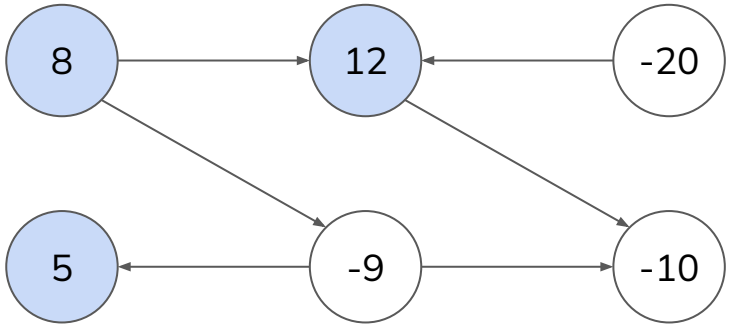
Modeling Problems with Cut

Let's apply "optimistic thinking":

What is the ***best ever scenario*** you can wish for?



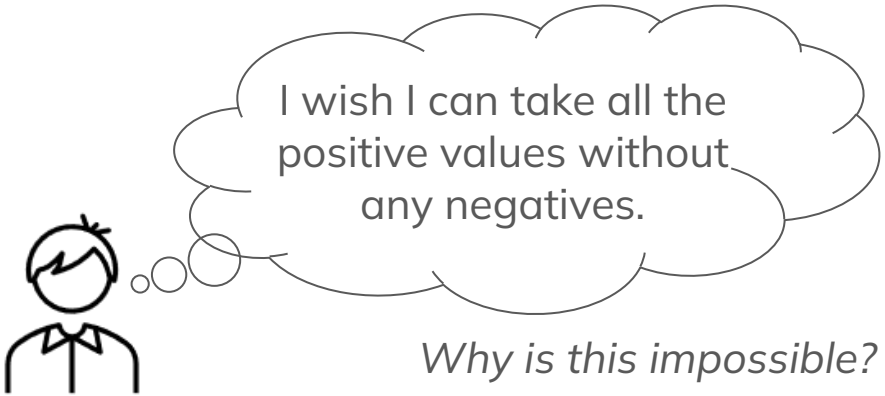
Why is this impossible?



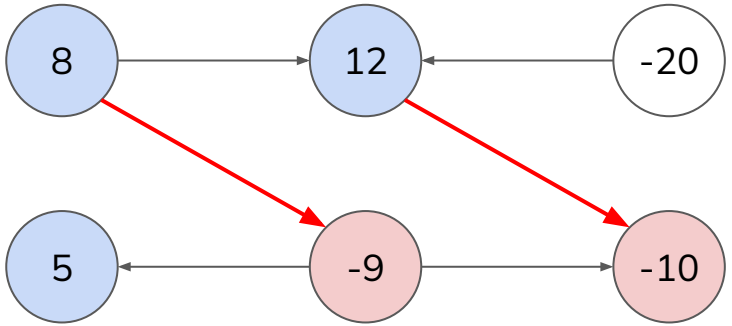
Modeling Problems with Cut

Let's apply "optimistic thinking":

What is the ***best ever scenario*** you can wish for?



Why is this impossible?

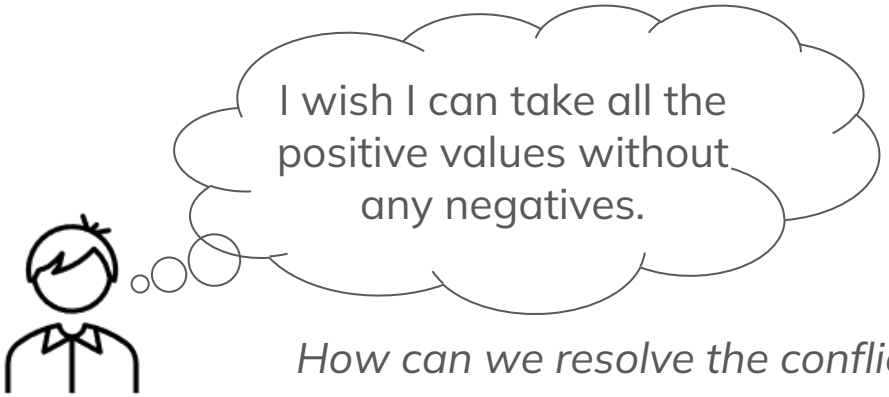


Conflicts: We picked u without picking v.
→ Express this as a **S-T path**.

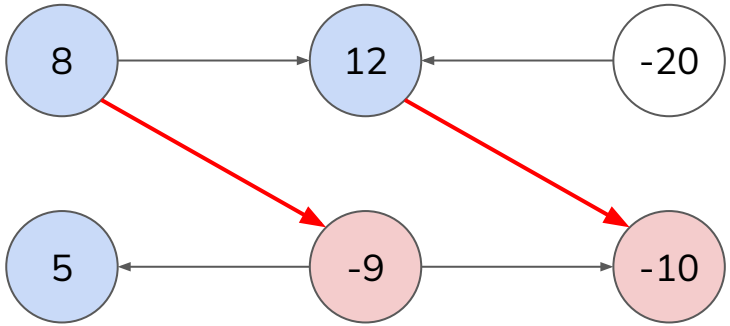
Modeling Problems with Cut

Let's apply "optimistic thinking":

What is the ***best ever scenario*** you can wish for?



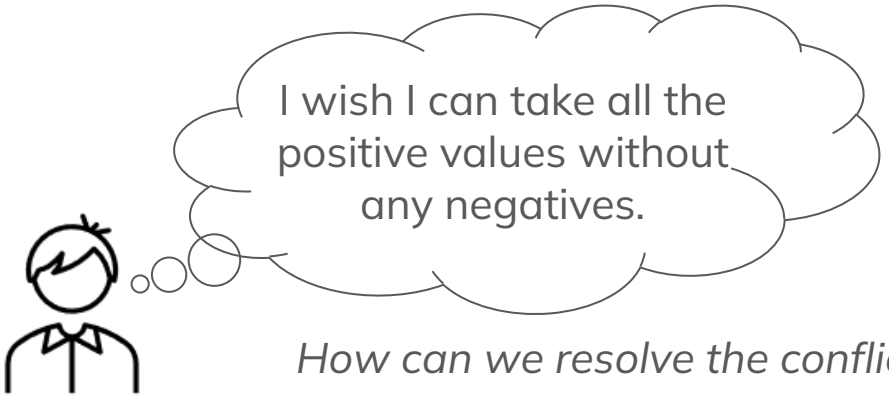
How can we resolve the conflict?



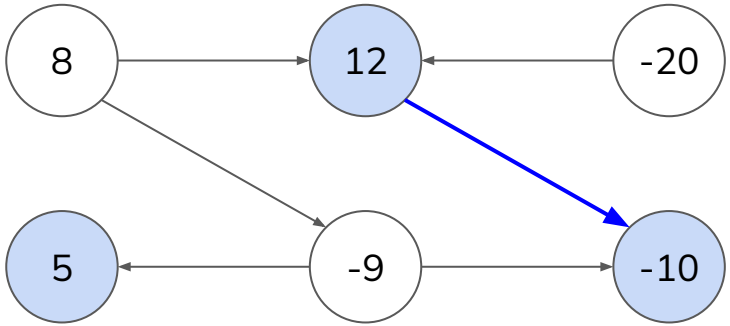
Modeling Problems with Cut

Let's apply "optimistic thinking":

What is the ***best ever scenario*** you can wish for?



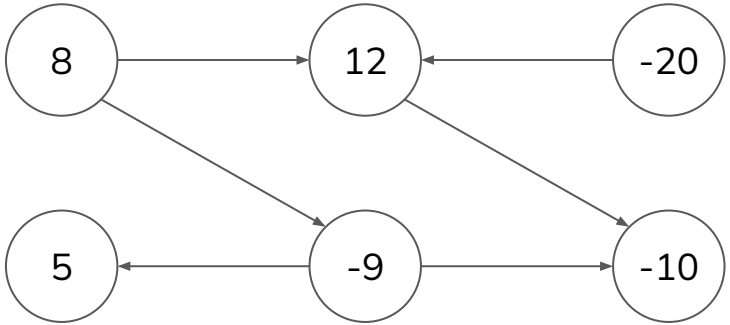
How can we resolve the conflict?

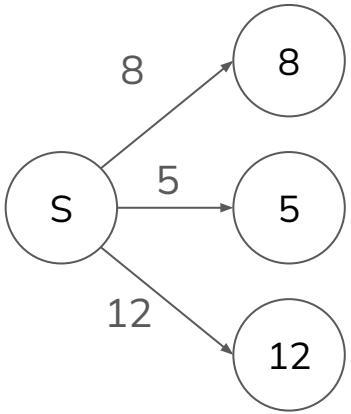


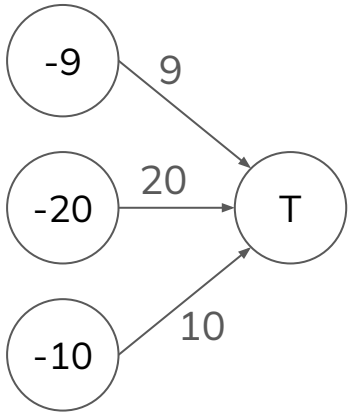
Action: Decide not to pick the positive vertex // Decide to pick the negative vertex.

Modeling Problems with Cut

- Create a vertex in the flow network for each vertex in the graph.
- By default, we choose the positive vertices but not the negative ones:
  - If we decide not to choose a positive vertex: pay a cost. (Build an edge S → u)
  - If we decide to choose a negative vertex: pay a cost. (Build an edge u → T)

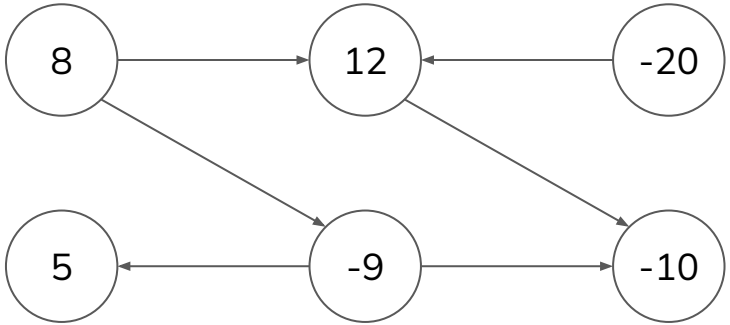


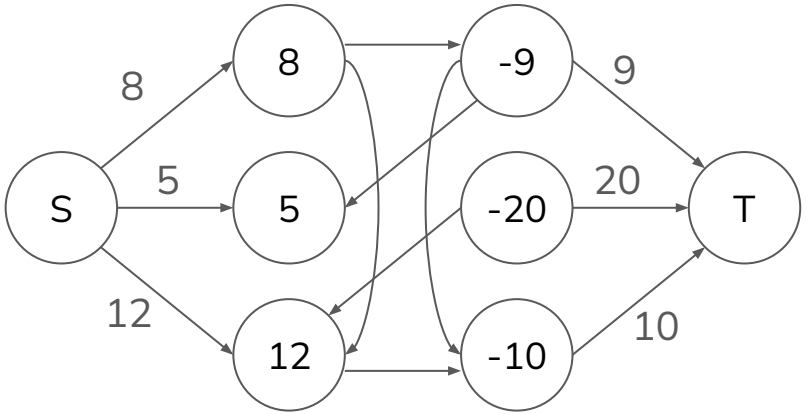




Modeling Problems with Cut

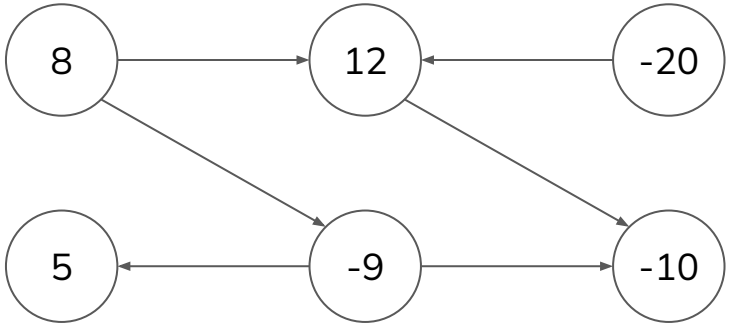
- A **conflict** happens if we picked u but did not pick v for some edge u → v.
  - This means the edges S → u (we picked u) and v → T exist (we did not pick v).
  - Build an edge u → v to form a S-T path (S → u → v → T).
  - What is the cost to "cut" this conflict?

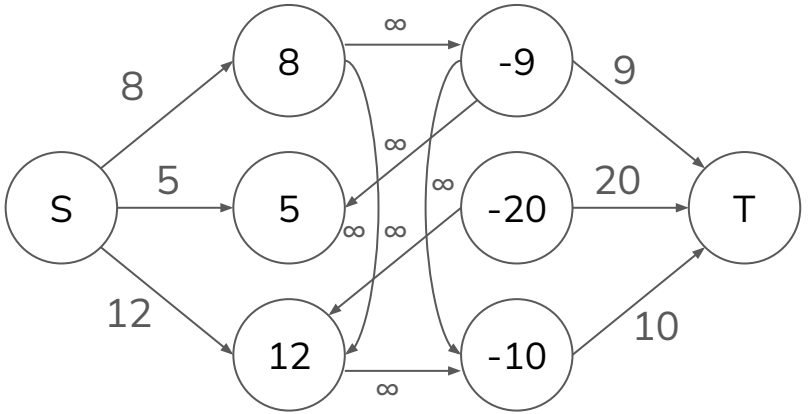




Modeling Problems with Cut

- A **conflict** happens if we picked u but did not pick v for some edge u → v.
  - This means the edges S → u (we picked u) and v → T exist (we did not pick v).
  - Build an edge u → v to form a S-T path (S → u → v → T).
  - What is the cost to "cut" this conflict? Infinity, it's illegal to cut this edge.

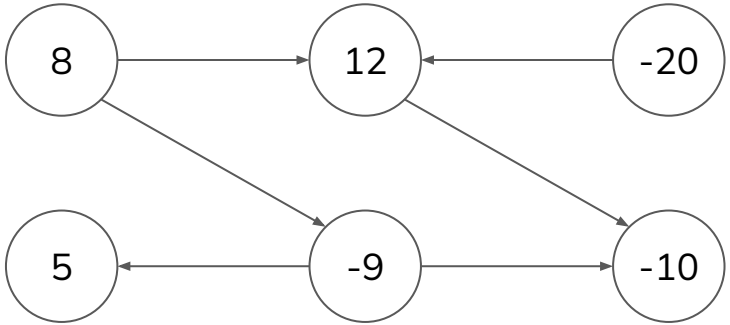


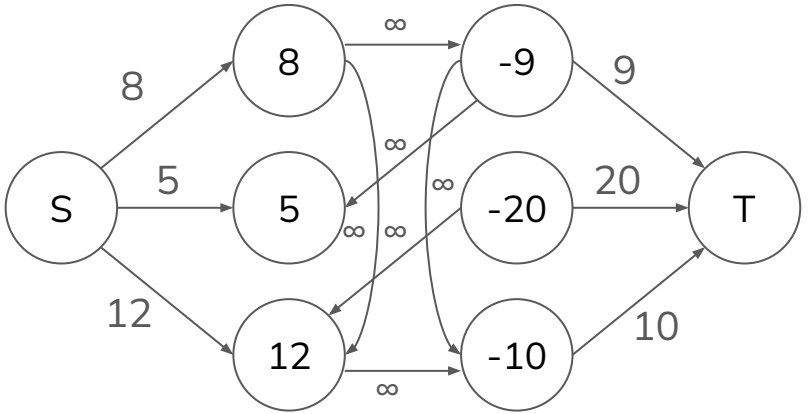


Modeling Problems with Cut

Solution:

- For each vertex with weight w, build the edge S → u with capacity w if w is positive, or build the edge u → T with capacity -w if w is negative.
- For each edge u → v, build the edge u → v with infinite capacity.
- Answer = Minimum Cut = Maximum Flow.





Modeling Problems with Cut

Project Selection Problem: You are given `N` projects and `M` machines. Each project requires a certain set of machines. Project `i`, if completed, generates `a[i]` revenue but machine `j` costs `b[j]` to purchase (it can be used **multiple times**). Find the maximum profit you can make by choosing machines and projects wisely.

Project	Machines	Revenue
1	1, 2	10
2	2, 3	20
3	2, 3, 4	3

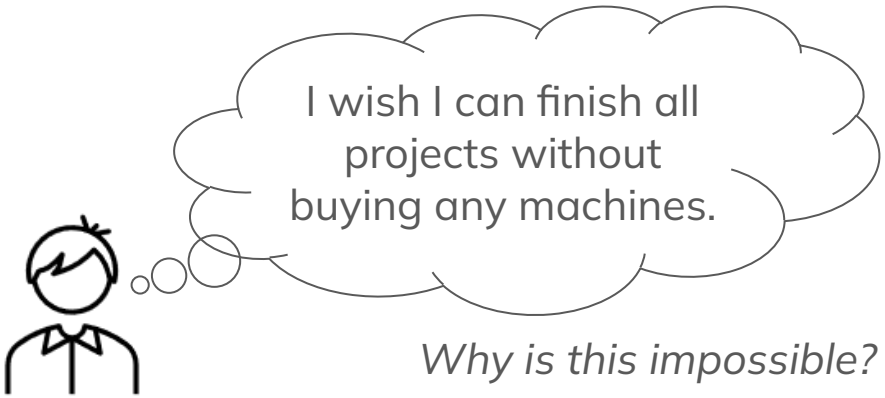
Machine	1	2	3	4
Cost	5	15	4	7

Profit = 10 + 20 - 5 - 15 - 4 = 6

Modeling Problems with Cut

Let's apply "optimistic thinking" again:

What is the ***best ever scenario*** you can wish for?



Why is this impossible?

Project	Machines	Revenue
1	1, 2	10
2	2, 3	20
3	2, 3, 4	3

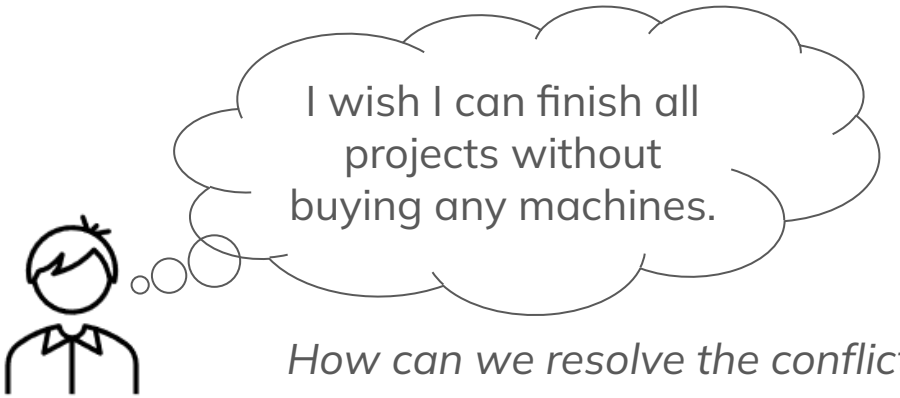
conflicts!

Machine	1	2	3	4
Cost	5	15	4	7

Modeling Problems with Cut

Let's apply "optimistic thinking" again:

What is the ***best ever scenario*** you can wish for?



How can we resolve the conflict?

Give up the revenue

Project	Machines	Revenue
1	1, 2	10
2	2, 3	20
3	2, 3, 4	3

Machine	1	2	3	4
Cost	5	15	4	7

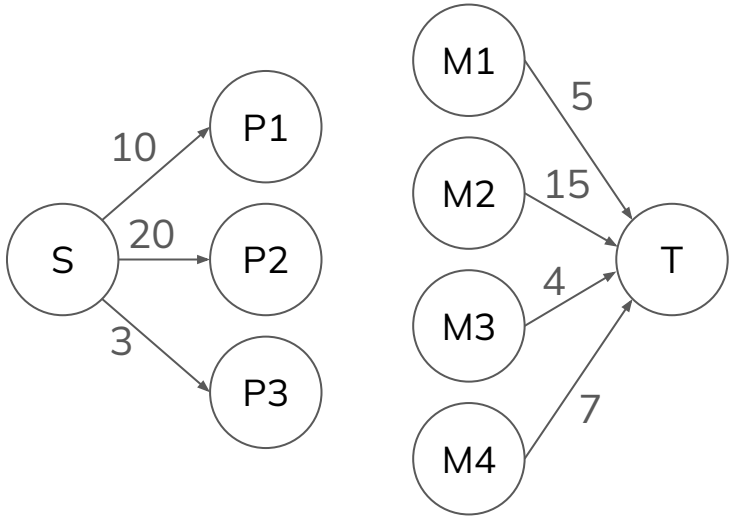
Pay for the machines

Modeling Problems with Cut

- Create a vertex in the flow network for each project and each machine.
- We can choose to abandon a project by paying a cost – create an edge S → p.
- We can choose to buy a machine by paying a cost – create an edge m → T.

Project	Machines	Revenue
1	1, 2	10
2	2, 3	20
3	2, 3, 4	3

Machine	1	2	3	4
Cost	5	15	4	7

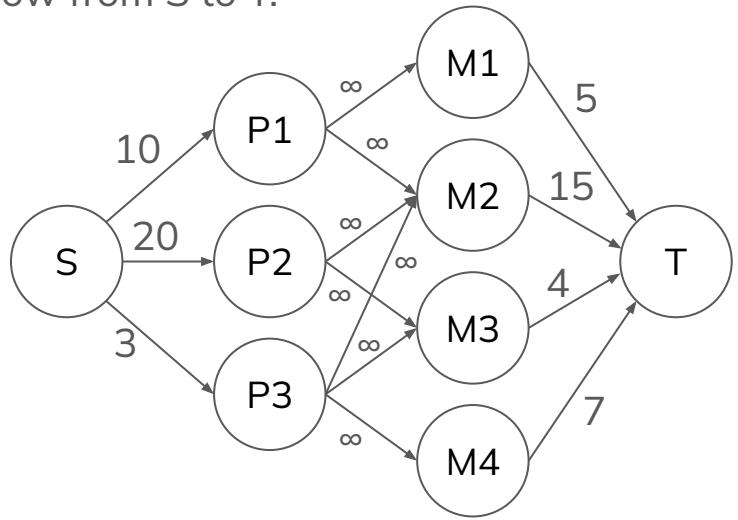


Modeling Problems with Cut

- A **conflict** happens if we chose a project without buying a machine required.
  - For each requirement (project, machine), we can build the edge p → m.
  - If we chose a project (the edge S → p remains) and did not buy the machine (the edge m → T remains), then there is a flow from S to T.

Project	Machines	Revenue
1	1, 2	10
2	2, 3	20
3	2, 3, 4	3

Machine	1	2	3	4
Cost	5	15	4	7



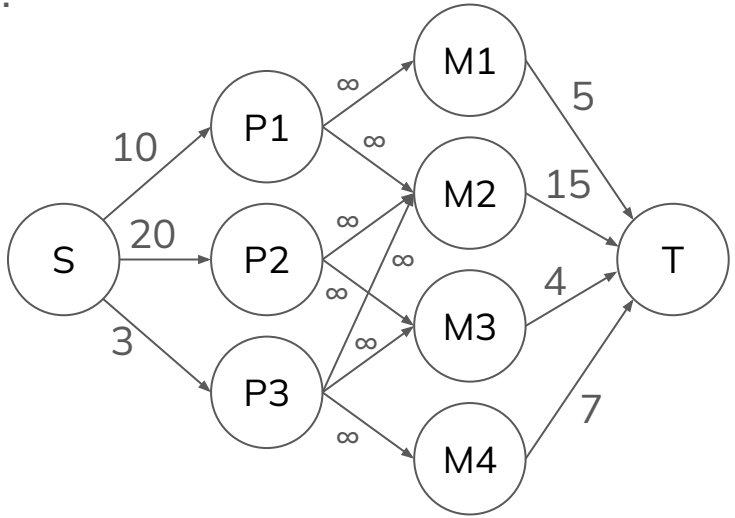
Modeling Problems with Cut

Solution:

- For each project p or machine m, build the edge S → p or m → T (capacity = cost).
- For each requirement (p, m), build the edge p → m with infinite capacity.
- Answer = Minimum Cut = Maximum Flow.

Project	Machines	Revenue
1	1, 2	10
2	2, 3	20
3	2, 3, 4	3

Machine	1	2	3	4
Cost	5	15	4	7



Summary

To solve **minimum cut** problems:

- Apply “optimistic thinking” – what is the ***best ever scenario*** you can get?
- Identify the **conflicts** and the **actions** you can do to resolve these conflicts.
- Model these conflicts/actions (edges) and costs (capacities) into a graph.
- Using **Max-Flow Min-Cut Theorem**, we can run Ford–Fulkerson Algorithm to find the maximum flow (= minimum cut).

Team Problem Set

Team Problem Set

- Try to work on graph modeling!
  - Teams of 3, work collaboratively!
  - Duration: Around 1 hour (depends on lesson time)

- Work on the same question *together*, as opposed to divide & conquer strategies.
  - Time given should be more than enough.
  - Team members should understand each other's solutions.

- Draw the flow network + Describe how to obtain the answer in 1 sentence.

- Teams will be invited to present their solutions to the whole class.