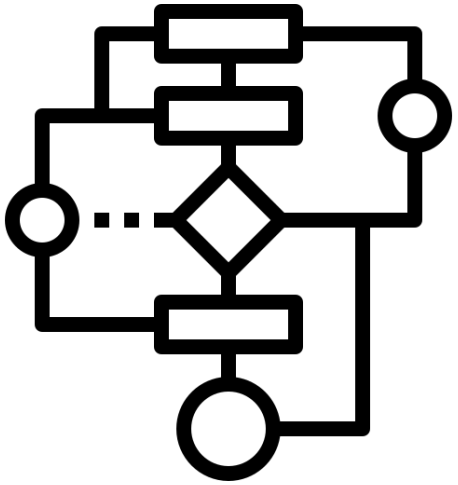


CS 310: Algorithms

Lecture 21

Instructor: Naveed Anwar Bhatti

Few Slides taken from Dr. Imdad's CS 510 course

Administrivia

- **Assignment 4** will be released on Wednesday
- **Assignment 1** and **2** marks will be released on Thursday
- **Assignment 3** marks will be released on 30th November

Quiz 5 – Question 1 Solution

In the class, we discussed the knapsack problem. Now, let's modify the rules slightly. You are allowed to split items into 1kg portions. Given the table below, devise the **MOST OPTIMAL ALGORITHM** (period) that can maximize the total value of items that can be placed in the knapsack.

Item	Value	Weight
1	**1**	**1**
2	**8**	**2**
3	**18**	**6**
4	**28**	**8**
5	**28**	**7**

C = 11

Quiz 5 – Question 1 Solution

In the class, we discussed the knapsack problem. Now, let's modify the rules slightly. You are allowed to split items into 1kg portions. Given the table below, devise the **MOST OPTIMAL ALGORITHM** (period) that can maximize the total value of items that can be placed in the knapsack.

Item	Value	Weight	Ratio
1	**1**	**1**	**1**
2	**8**	**2**	**4**
3	**18**	**6**	**3**
4	**28**	**8**	**3.5**
5	**28**	**7**	**4**

C = 11

Quiz 5 – Question 1 Solution

In the class, we discussed the knapsack problem. Now, let's modify the rules slightly. You are allowed to split items into 1kg portions. Given the table below, devise the **MOST OPTIMAL ALGORITHM** (period) that can maximize the total value of items that can be placed in the knapsack.

Item	Value	Weight	Ratio
1	**1**	**1**	**1**
3	**18**	**6**	**3**
4	**28**	**8**	**3.5**
2	**8**	**2**	**4**
5	**28**	**7**	**4**

C = 11

Quiz 5 – Question 1 Solution

In the class, we discussed the knapsack problem. Now, let's modify the rules slightly. You are allowed to split items into 1kg portions. Given the table below, devise the **MOST OPTIMAL ALGORITHM** (period) that can maximize the total value of items that can be placed in the knapsack.

Item	Value	Weight	Ratio
1	**1**	**1**	**1**
3	**18**	**6**	**3**
4	**28**	**8**	**3.5**
2	**8**	**2**	**4**
5	**28**	**7**	**4**

C = 4

28

7

Quiz 5 – Question 1 Solution

In the class, we discussed the knapsack problem. Now, let's modify the rules slightly. You are allowed to split items into 1kg portions. Given the table below, devise the **MOST OPTIMAL ALGORITHM** (period) that can maximize the total value of items that can be placed in the knapsack.

Item	Value	Weight	Ratio
1	1	1	1
3	18	6	3
4	28	8	3.5
2	8	2	4
5	28	7	4

C = 2

28 8

7 + 2

Quiz 5 – Question 1 Solution

In the class, we discussed the knapsack problem. Now, let's modify the rules slightly. You are allowed to split items into 1kg portions. Given the table below, devise the **MOST OPTIMAL ALGORITHM** (period) that can maximize the total value of items that can be placed in the knapsack.

Item	Value	Weight	Ratio
1	1	1	1
3	18	6	3
4	28	8	3.5
2	8	2	4
5	28	7	4

C = 0

28 8 (2x3.5)

7 + 2 + 2

Quiz 5 – Question 1 Solution

In the class, we discussed the knapsack problem. Now, let's modify the rules slightly. You are allowed to split items into 1kg portions. Given the table below, devise the **MOST OPTIMAL ALGORITHM** (period) that can maximize the total value of items that can be placed in the knapsack.

Item	Value	Weight	Ratio
1	1	1	1
3	18	6	3
4	28	8	3.5
2	8	2	4
5	28	7	4

C = 0

28 8 (2x3.5) = 43

7 + 2 + 2

Quiz 5 – Question 1 Solution

In the class, we discussed the knapsack problem. Now, let's modify the rules slightly. You are allowed to split items into 1kg portions. Given the table below, devise the **MOST OPTIMAL ALGORITHM** (period) that can maximize the total value of items that can be placed in the knapsack.

```
// List of items, each with a value and a weight
items = [(value1, weight1), (value2, weight2), ..., (valueN, weightN)]

sort items by (value/weight) in descending order

capacity = 11
total_value = 0

for item in items:
    if capacity > 0:
        take_weight = min(item.weight, capacity)
        total_value += take_weight * (item.value / item.weight)
        capacity -= take_weight
    else:
        break
```

O(n*log*n)

O(n)

Time Complexity

O(n*log*n)

Quiz 5 – Question 2 Solution

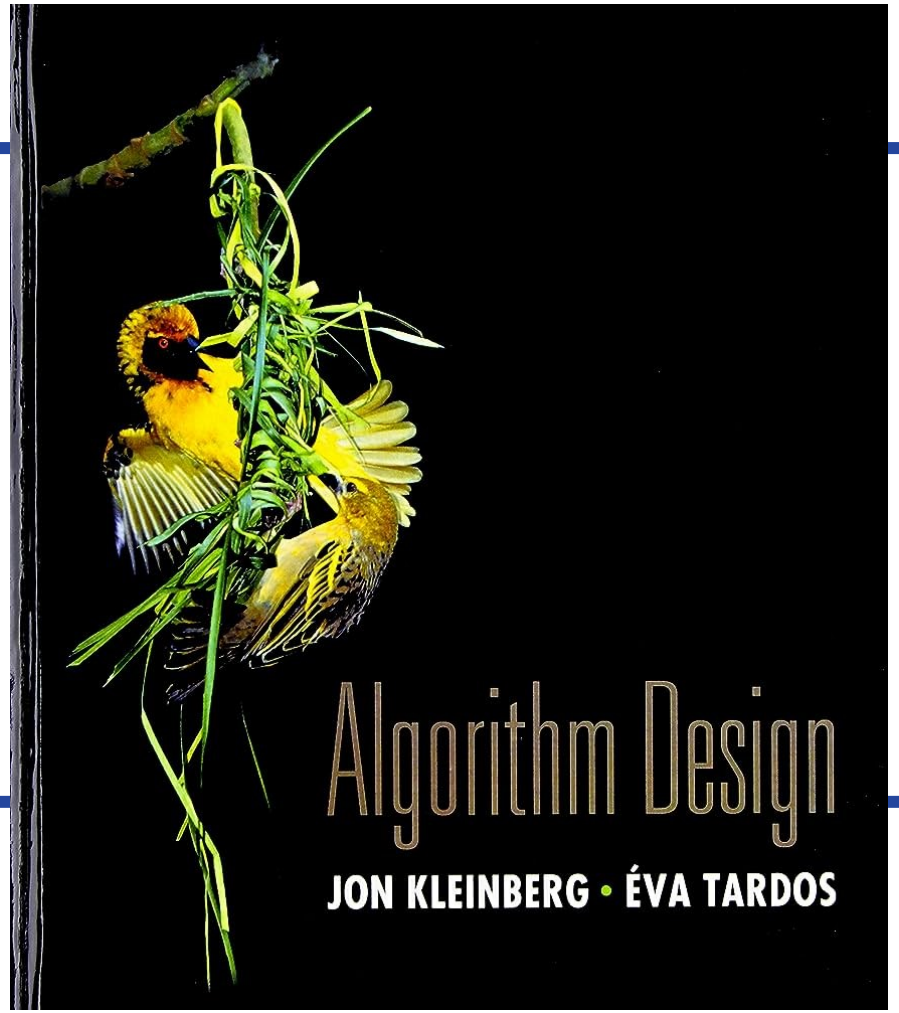
In the class, we discussed **Segmented Least Squares** problem using dynamic programming which has the running time complexity of **$O(n^3)$ (shown below). How can we improve it? You don't need to write the pseudo, just explain it in few lines**.

```
INPUT: n, p1,…,pN , c

for i=0 to n:
     M[n]= -1
M[0]= 0

OPT(){
    if n == 0:
        return 0

    if M[0]!=-1:
        return M[n]

    min_cost = 0;

    for i=1 to n:                    O(n)
        cost = e(i,n) + c + OPT(i - 1)
        if cost < min_cost:
            min_cost = cost

    M[n] = min_cost
    return min_cost
}
```

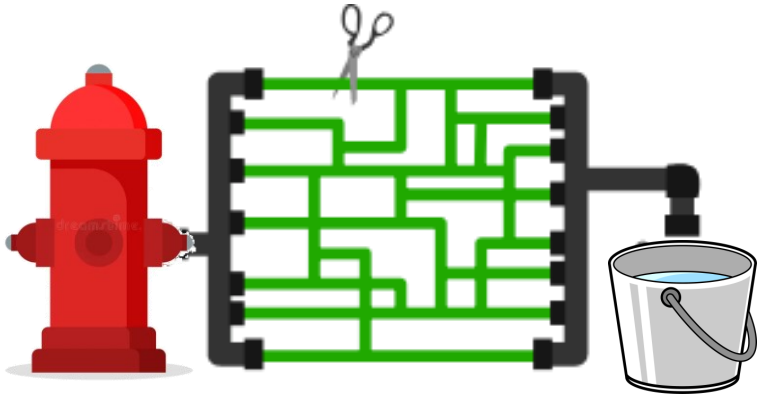
O(n) — O(n)

O(n)

Pre-compute least squares error e(i,n) for every possible segment, which takes $O(n^2)$ and save it into matrix. This would allow later e(i, n) to be calculated in constant time using pre-stored calculations, reducing the overall time complexity to $O(n^2)$



Chapter 7:
Network Flow



Section :
Maximum Flow – Min Cut Problem

Flow Networks

A network of pipelines along which water can be sent

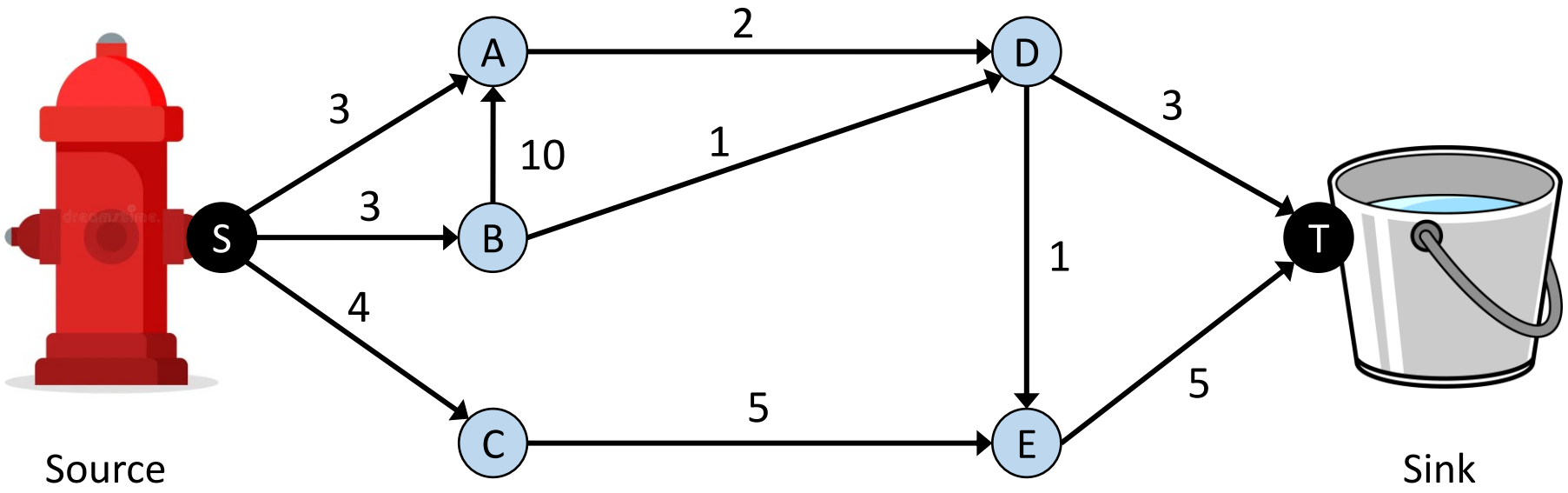
Source

Sink

Flow Networks

A network of pipelines along which water can be sent

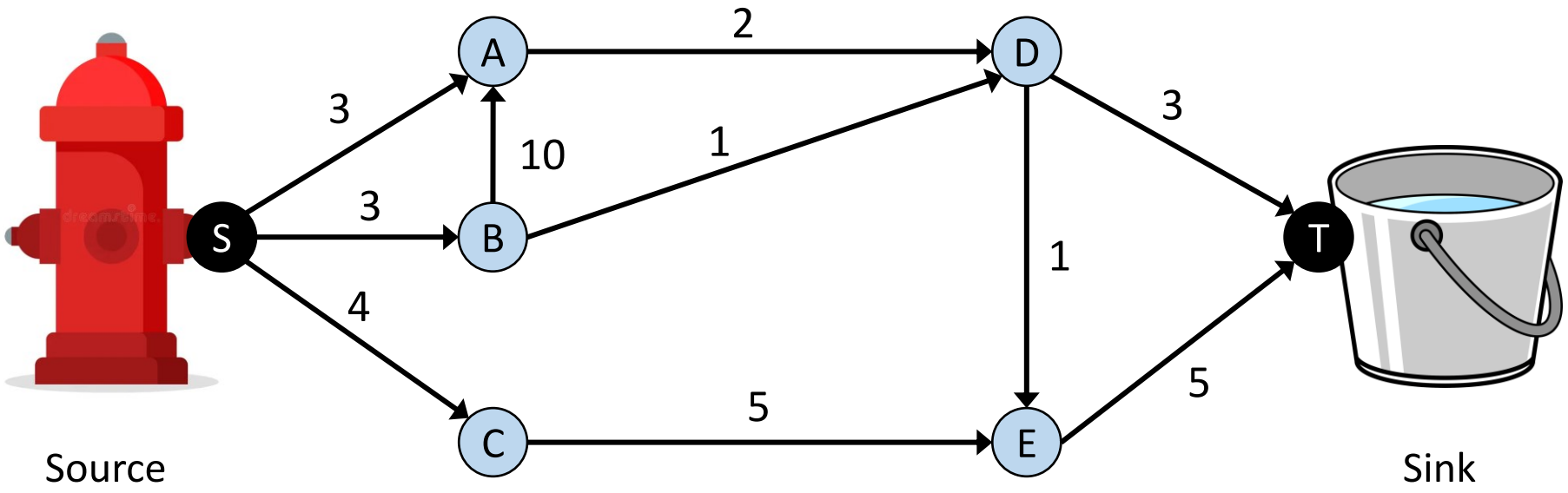
The **goal** is to flow as much water from **S** to **T** as possible



Flow Networks

A network of pipelines along which water can be sent

The **goal** is to flow as much water from **S** to **T** as possible



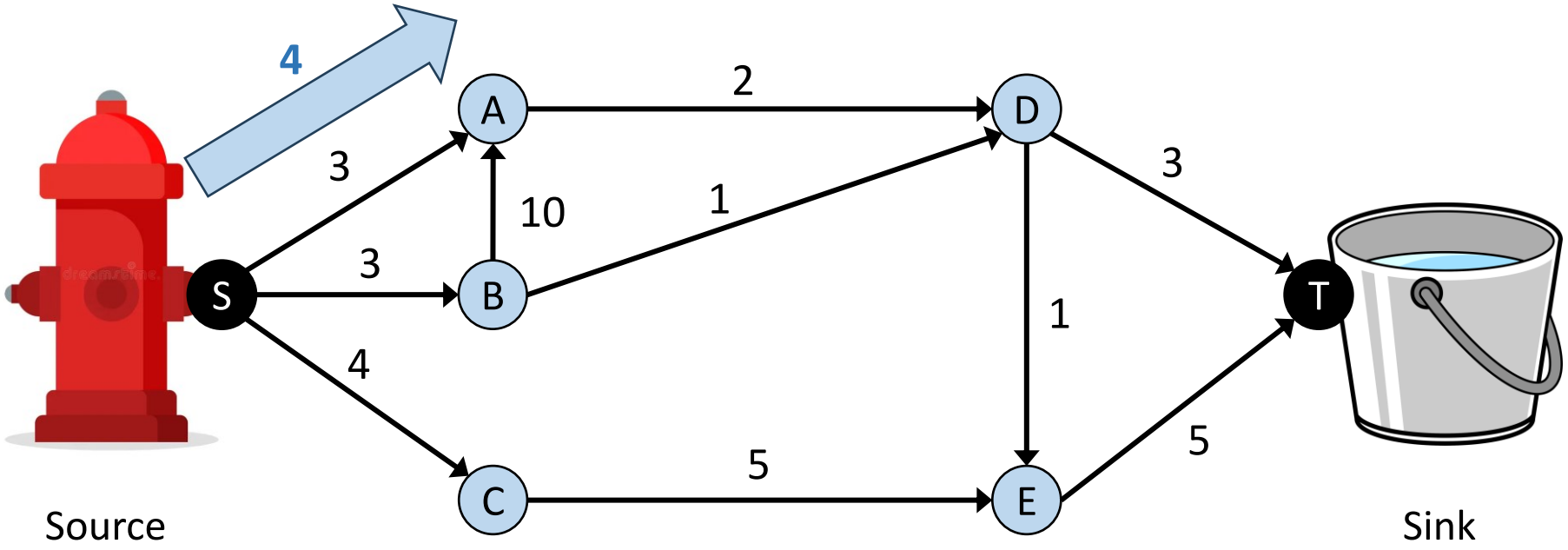
There are TWO restrictions:

- A pipeline cannot carry more water than the weight of the corresponding edge
- No vertex can store any water

Flow Networks

A network of pipelines along which water can be sent

The **goal** is to flow as much water from **S** to **T** as possible



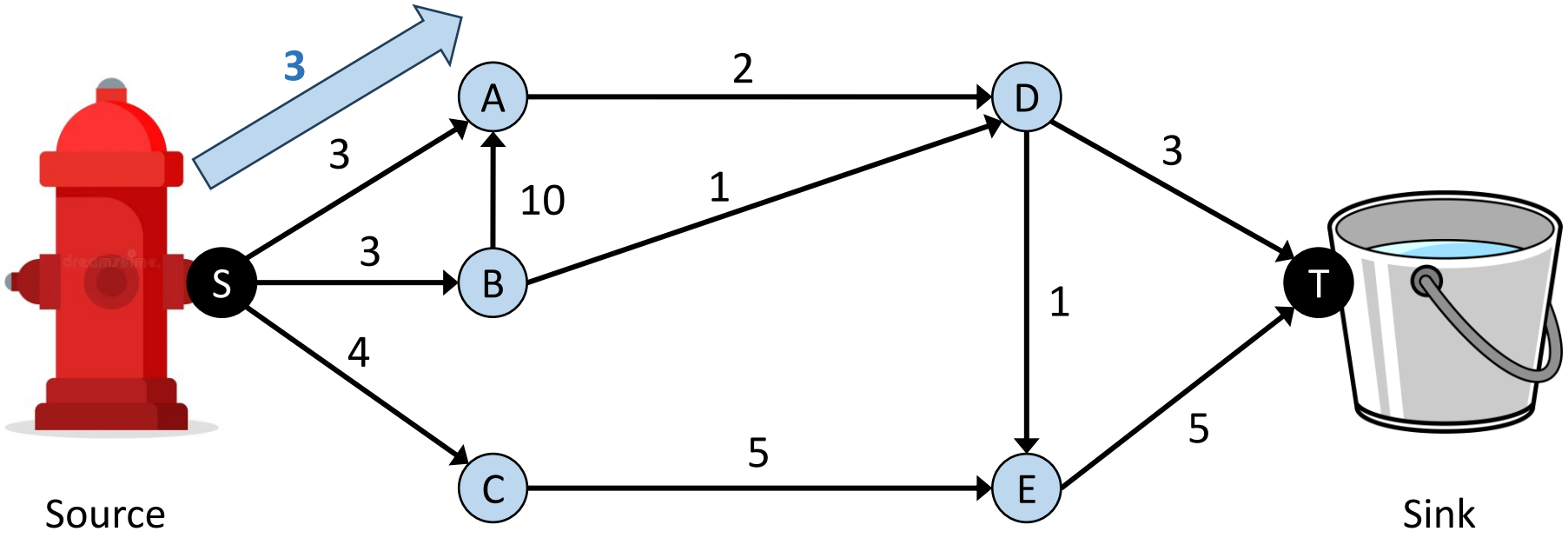
There are TWO restrictions:

- A pipeline cannot carry more water than the weight of the corresponding edge
- No vertex can store any water

Flow Networks

A network of pipelines along which water can be sent

The **goal** is to flow as much water from **S** to **T** as possible



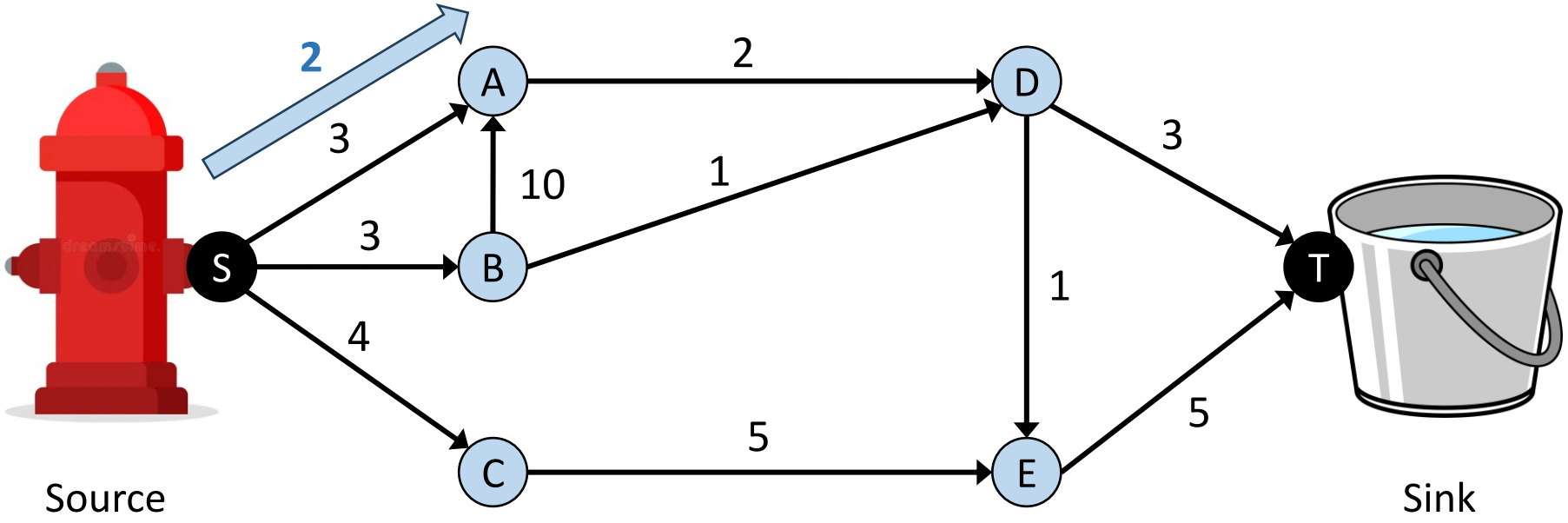
There are TWO restrictions:

- A pipeline cannot carry more water than the weight of the corresponding edge
- No vertex can store any water

Flow Networks

A network of pipelines along which water can be sent

The **goal** is to flow as much water from **S** to **T** as possible



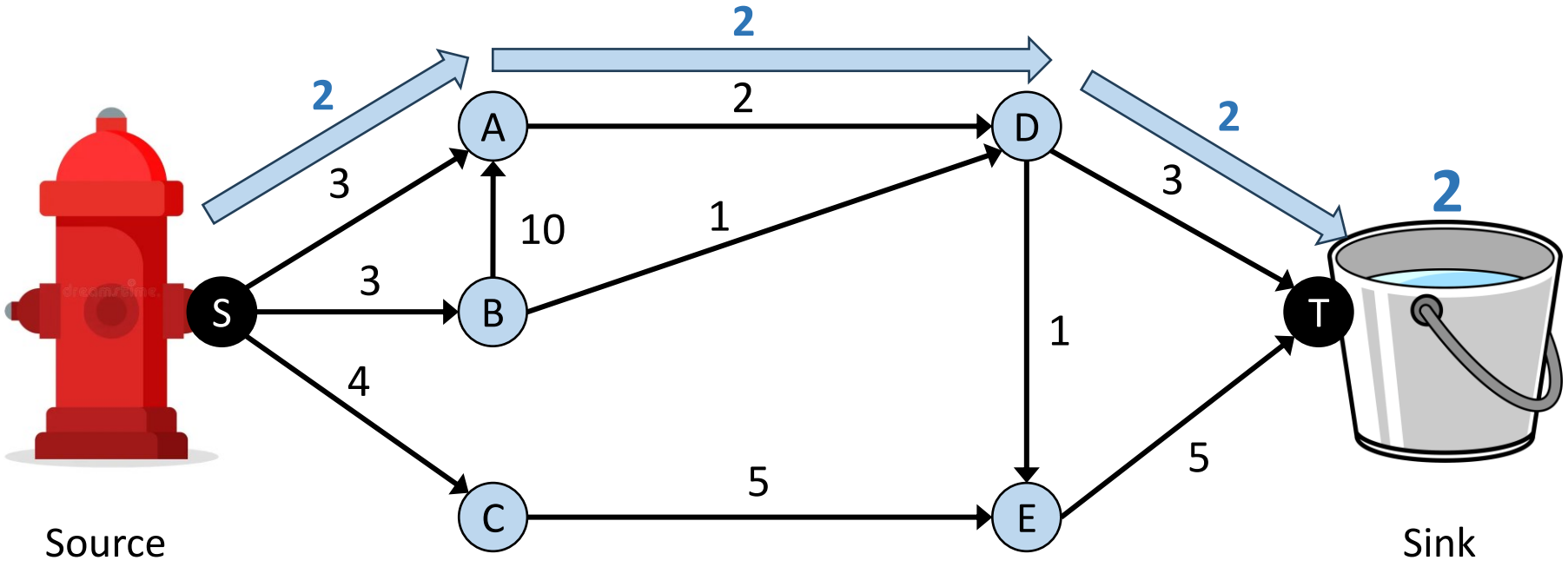
There are TWO restrictions:

- A pipeline cannot carry more water than the weight of the corresponding edge
- No vertex can store any water

Flow Networks

A network of pipelines along which water can be sent

The **goal** is to flow as much water from **S** to **T** as possible



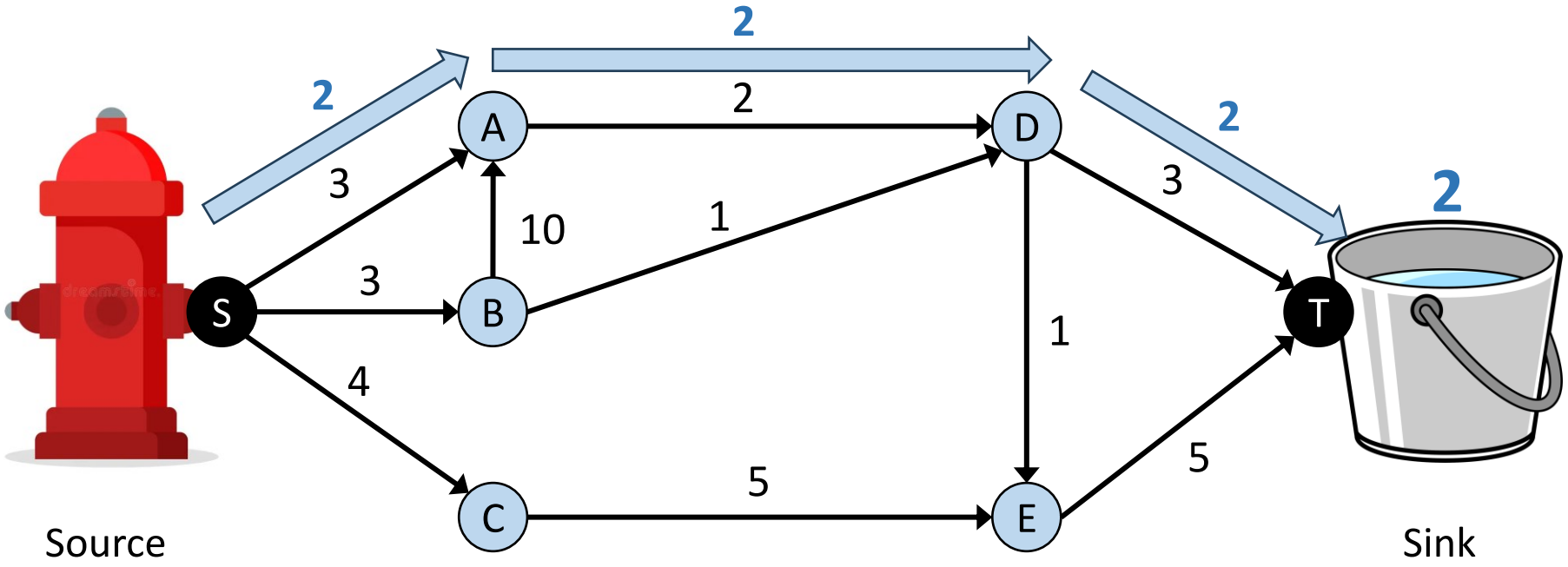
There are TWO restrictions:

- A pipeline cannot carry more water than the weight of the corresponding edge
- No vertex can store any water

Flow Networks

A network of pipelines along which water can be sent

The **goal** is to flow as much water from **S** to **T** as possible

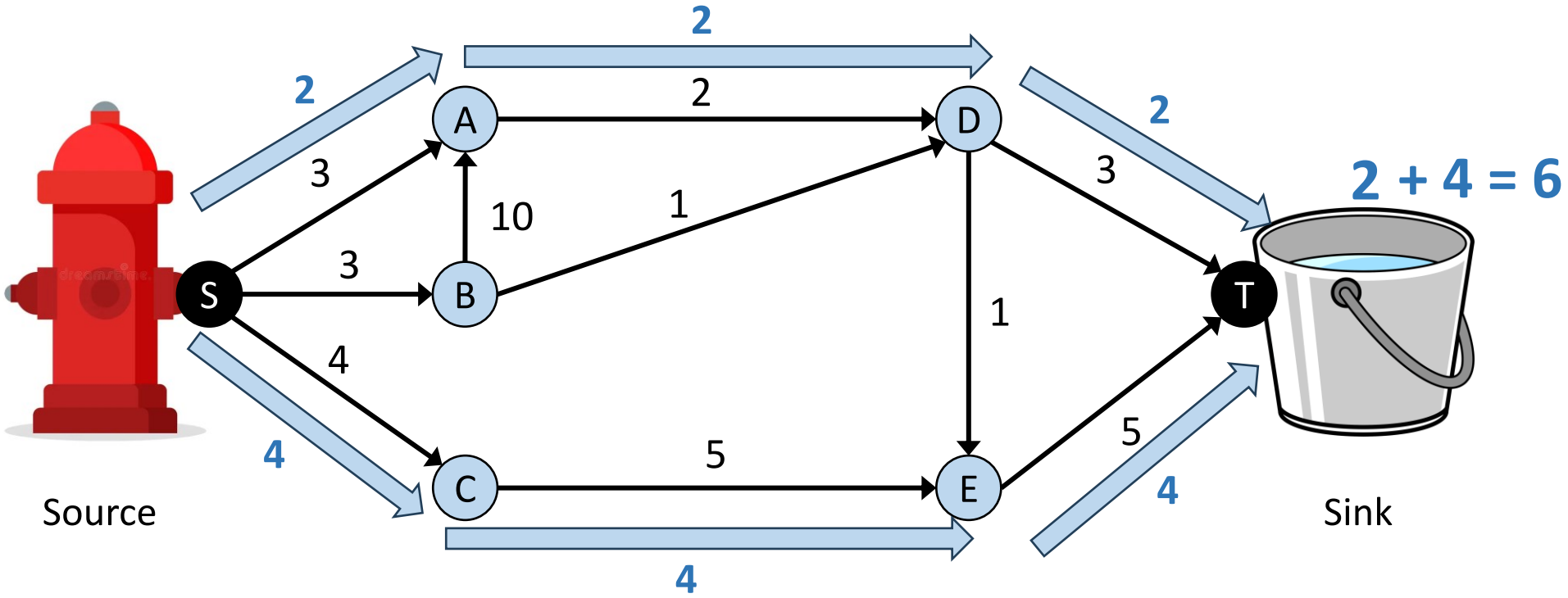


Is this the **best flow** (the largest amount of water that can be supplied)?

Flow Networks

A network of pipelines along which water can be sent

The **goal** is to flow as much water from **S** to **T** as possible

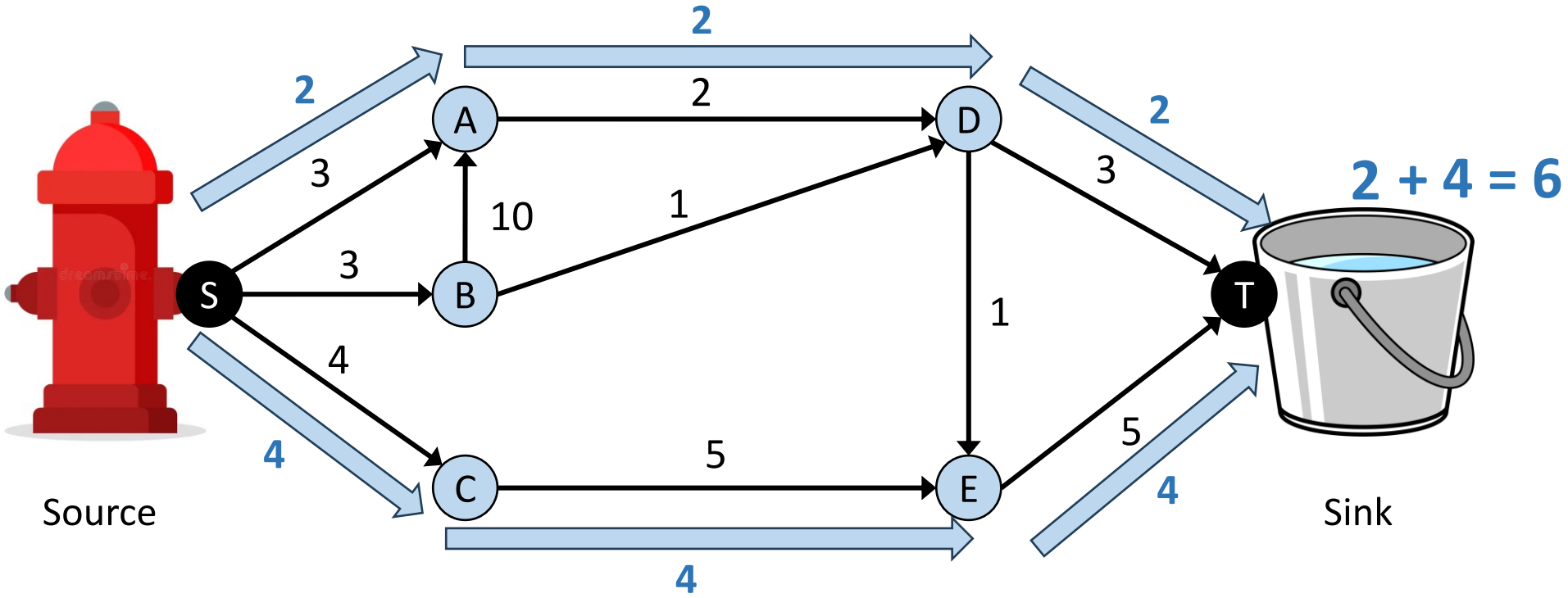


Is this the **best flow** (the largest amount of water that can be supplied)?

Flow Networks

A network of pipelines along which water can be sent

The **goal** is to flow as much water from **S** to **T** as possible



Is this the **best flow** (the largest amount of water that can be supplied)?

How do we determine that a given flow is the **maximum possible**?

Max Flow – Problem Formulation

A Flow network: A directed graph with weights on edges

- Models a transportation network
- Edges can carry traffic and nodes switch traffic between edges
  - Highway network: vertices: intersections/interchanges, edges: roads
  - Communication network: vertices: switches/routers, edges: links
  - Fluid networks: vertices: junctures where pipes are plugged, edges: pipelines

Edge weights are capacities of edges (max traffic they can carry)

Max Flow – Problem Formulation

A Flow network: A directed graph with weights on edges

- Models a transportation network
- Edges can carry traffic and nodes switch traffic between edges
  - Highway network: vertices: intersections/interchanges, edges: roads
  - Communication network: vertices: switches/routers, edges: links
  - Fluid networks: vertices: junctures where pipes are plugged, edges: pipelines

Edge weights are capacities of edges (max traffic they can carry)

(capacity of the edge uv): $e = uv$ is associated with $c_e = c_{uv} \in \mathbb{R}^+$

Max Flow – Problem Formulation

A Flow network: A directed graph with weights on edges

- Models a transportation network
- Edges can carry traffic and nodes switch traffic between edges
  - Highway network: vertices: intersections/interchanges, edges: roads
  - Communication network: vertices: switches/routers, edges: links
  - Fluid networks: vertices: junctures where pipes are plugged, edges: pipelines

Edge weights are capacities of edges (max traffic they can carry)

(capacity of the edge uv): $e = uv$ is associated with $c_e = c_{uv} \in \mathbb{R}^+$

(source s): $deg^-(s) = 0$ is the traffic generator

Max Flow – Problem Formulation

A Flow network: A directed graph with weights on edges

- Models a transportation network
- Edges can carry traffic and nodes switch traffic between edges
  - Highway network: vertices: intersections/interchanges, edges: roads
  - Communication network: vertices: switches/routers, edges: links
  - Fluid networks: vertices: junctures where pipes are plugged, edges: pipelines

Edge weights are capacities of edges (max traffic they can carry)

(capacity of the edge uv): $e = uv$ is associated with $c_e = c_{uv} \in \mathbb{R}^+$

(source s): $deg^-(s) = 0$ is the traffic generator

(sink t): $deg^+(t) = 0$ is the traffic consumer

Max Flow – Problem Formulation

A Flow network: A directed graph with weights on edges

- Models a transportation network
- Edges can carry traffic and nodes switch traffic between edges
  - Highway network: vertices: intersections/interchanges, edges: roads
  - Communication network: vertices: switches/routers, edges: links
  - Fluid networks: vertices: junctures where pipes are plugged, edges: pipelines

Edge weights are capacities of edges (max traffic they can carry)

(capacity of the edge uv): $e = uv$ is associated with $c_e = c_{uv} \in \mathbb{R}^+$

(source s): $deg^-(s) = 0$ is the traffic generator

(sink t): $deg^+(t) = 0$ is the traffic consumer

A $s - t$ flow is given by assigning each edge e of G a flow $f_e \in \mathbb{R}^+$ with $f_e \leq c_e$

Max Flow – Problem Formulation

A Flow network: A directed graph with weights on edges

- Models a transportation network
- Edges can carry traffic and nodes switch traffic between edges
  - Highway network: vertices: intersections/interchanges, edges: roads
  - Communication network: vertices: switches/routers, edges: links
  - Fluid networks: vertices: junctures where pipes are plugged, edges: pipelines

Edge weights are capacities of edges (max traffic they can carry)

(capacity of the edge uv): $e = uv$ is associated with $c_e = c_{uv} \in \mathbb{R}^+$

(source s): $deg^-(s) = 0$ is the traffic generator

(sink t): $deg^+(t) = 0$ is the traffic consumer

A $s - t$ flow is given by assigning each edge e of G a flow $f_e \in \mathbb{R}^+$ with $f_e \leq c_e$

flow $f : E \to \mathbb{R}^+$ satisfying the capacity and storage constraints

Max Flow – Problem Formulation

Given a flow network $G = (V, E, c)$, $c : E \to \mathbb{R}^+$
$s \in V$ a source and $t \in V$ a sink

Max Flow – Problem Formulation

Given a flow network $G = (V, E, c)$, $c : E \to \mathbb{R}^+$

$s \in V$ a source and $t \in V$ a sink

$f : E \to \mathbb{R}^+$ $(f_e = f(e))$ is a flow if it satisfies

Max Flow – Problem Formulation

Given a flow network $G = (V, E, c)$, $c : E \to \mathbb{R}^+$
$s \in V$ a source and $t \in V$ a sink
$f : E \to \mathbb{R}^+$ $(f_e = f(e))$ is a flow if it satisfies

1. (capacity constraints):

$$\forall\, e \in E \quad : \quad 0 \leq f_e \leq c_e$$

Max Flow – Problem Formulation

Given a flow network $G = (V, E, c)$, $c : E \to \mathbb{R}^+$
$s \in V$ a source and $t \in V$ a sink
$f : E \to \mathbb{R}^+$ $(f_e = f(e))$ is a flow if it satisfies

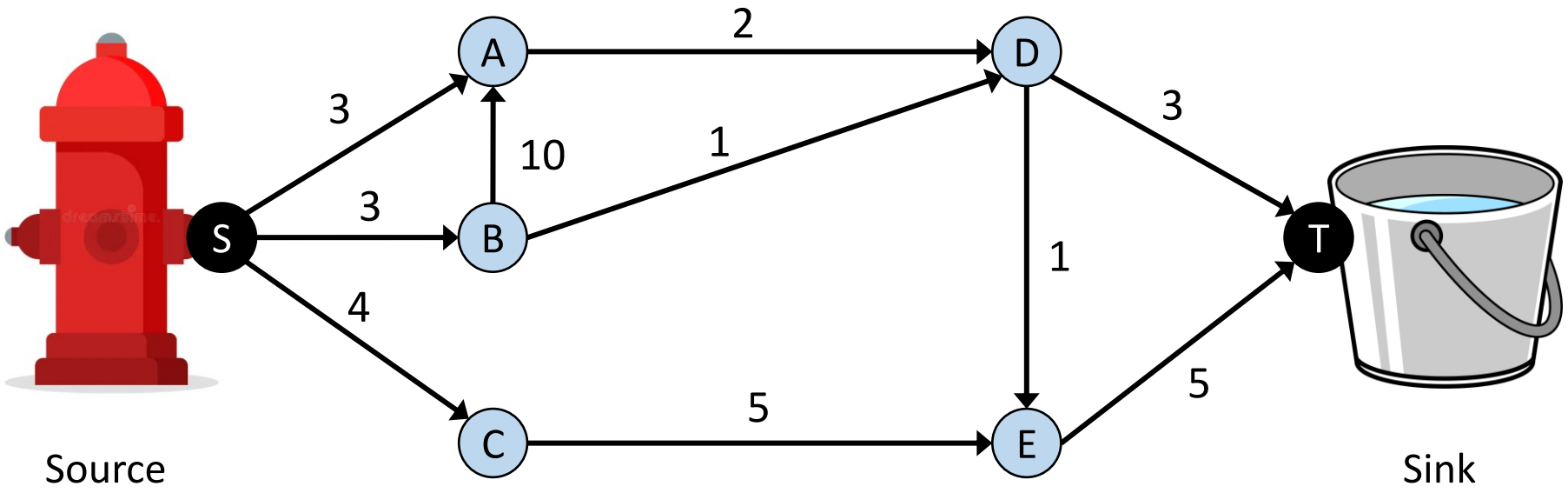
1. (capacity constraints):

$$\forall\, e \in E \quad : \quad 0 \leq f_e \leq c_e$$

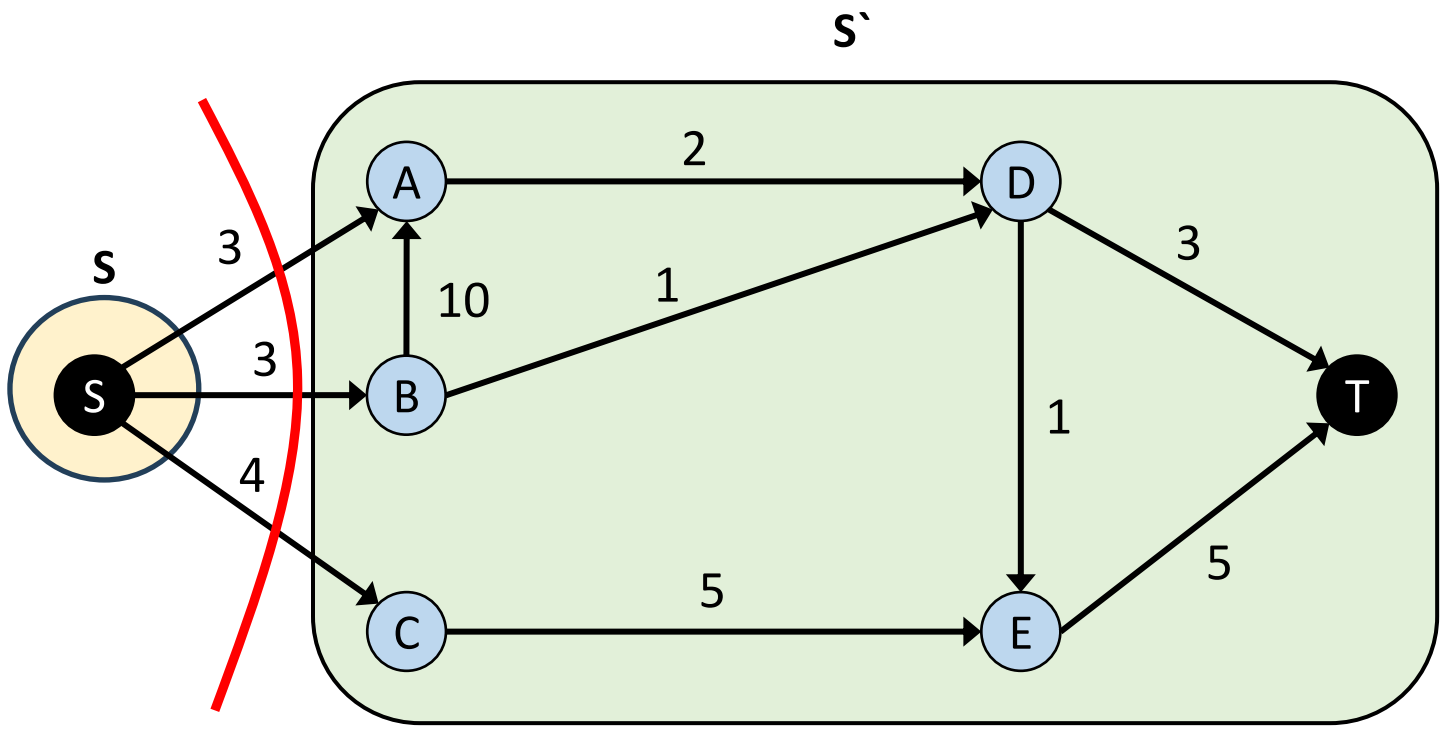
2. (flow conservation constraints):

$$\forall\, v \in V\,, v \neq s, t \qquad \underbrace{\sum_{e \text{ into } v} f_e}_{\text{total flow incoming to } v} = \underbrace{\sum_{e \text{ out of } v} f_e}_{\text{total flow outgoing from } v}$$

Max Flow – Min Cut

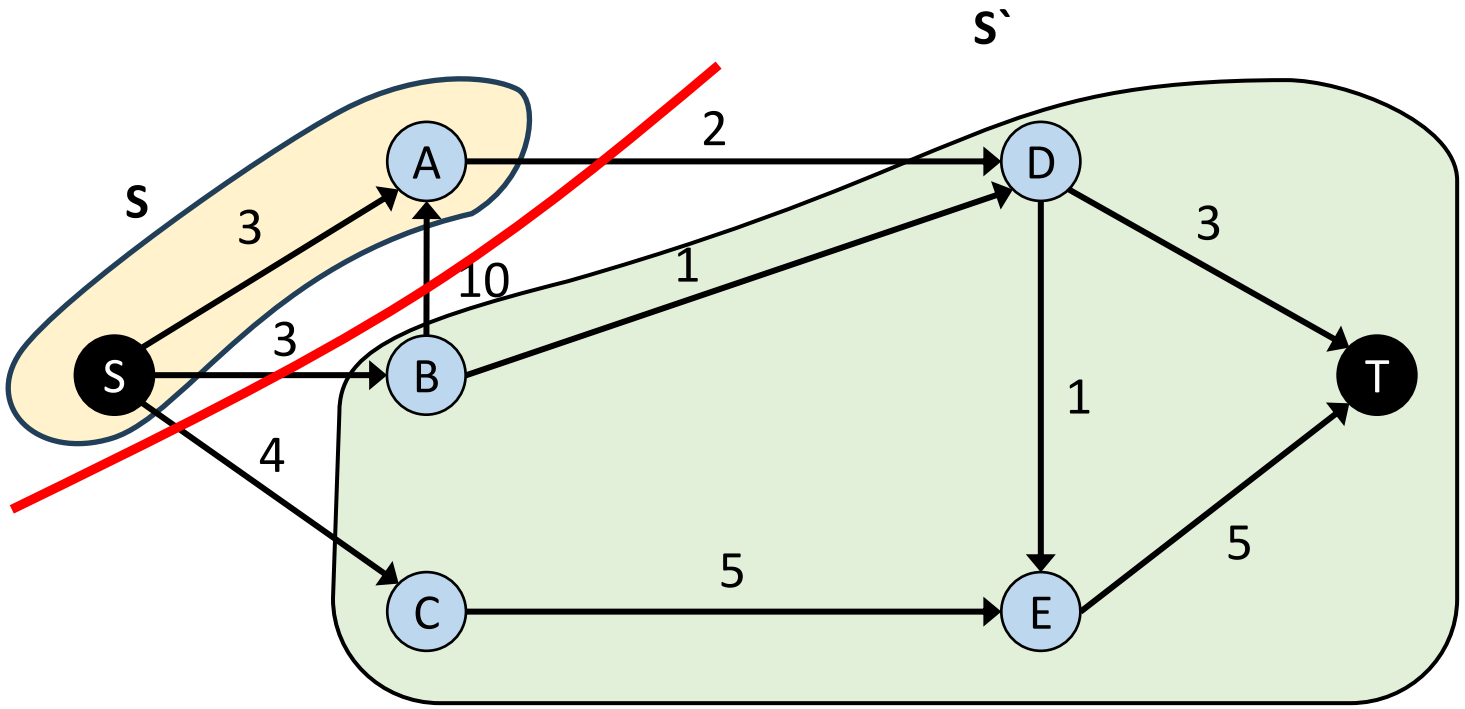


Max Flow – Min Cut (s-t cut)



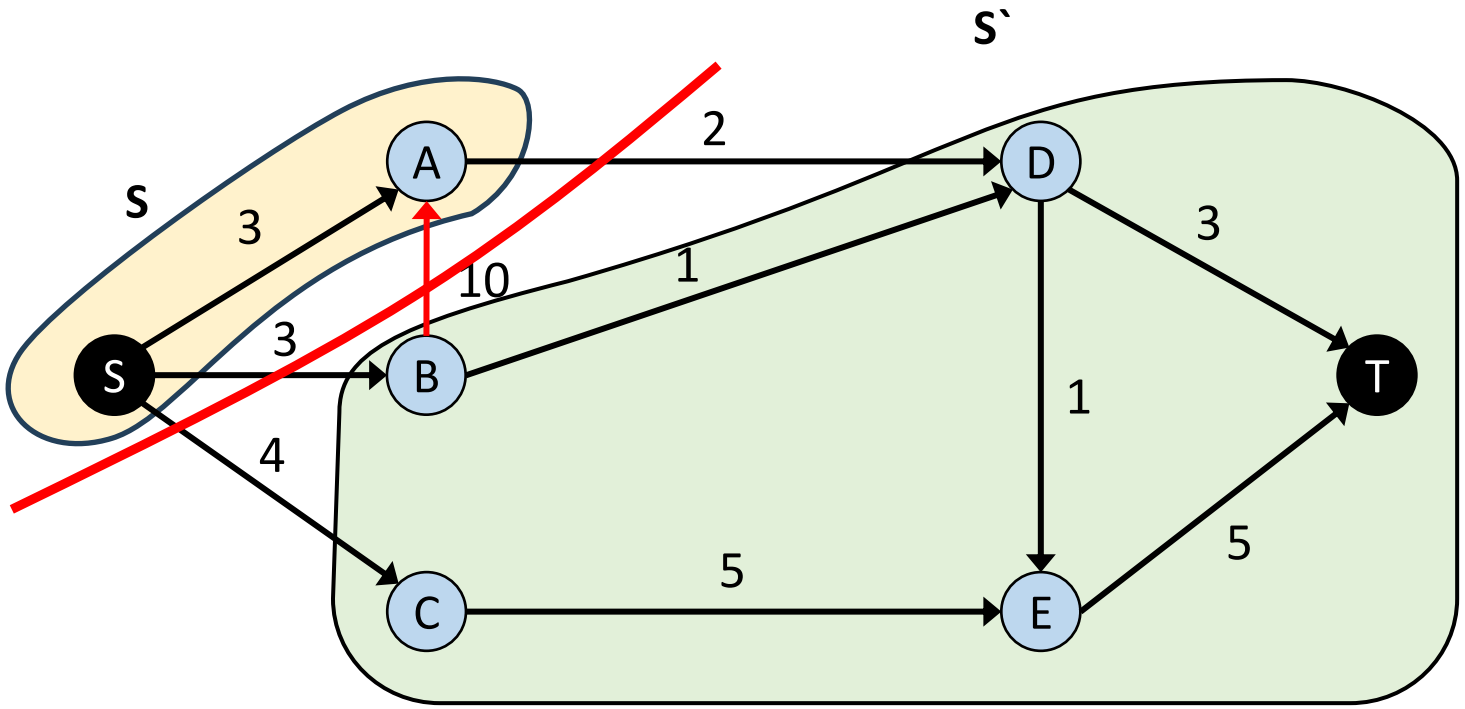
10

Max Flow – Min Cut (s-t cut)



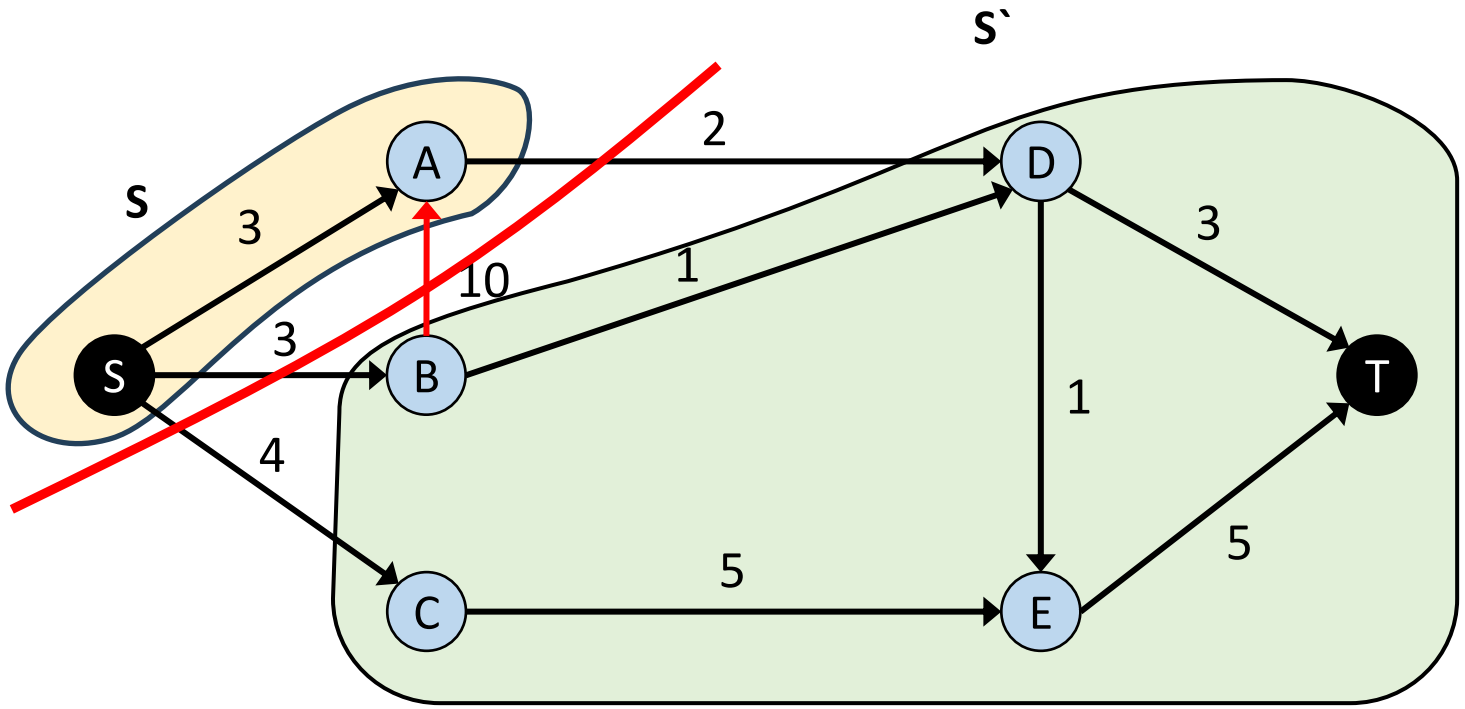
10

Max Flow – Min Cut (s-t cut)

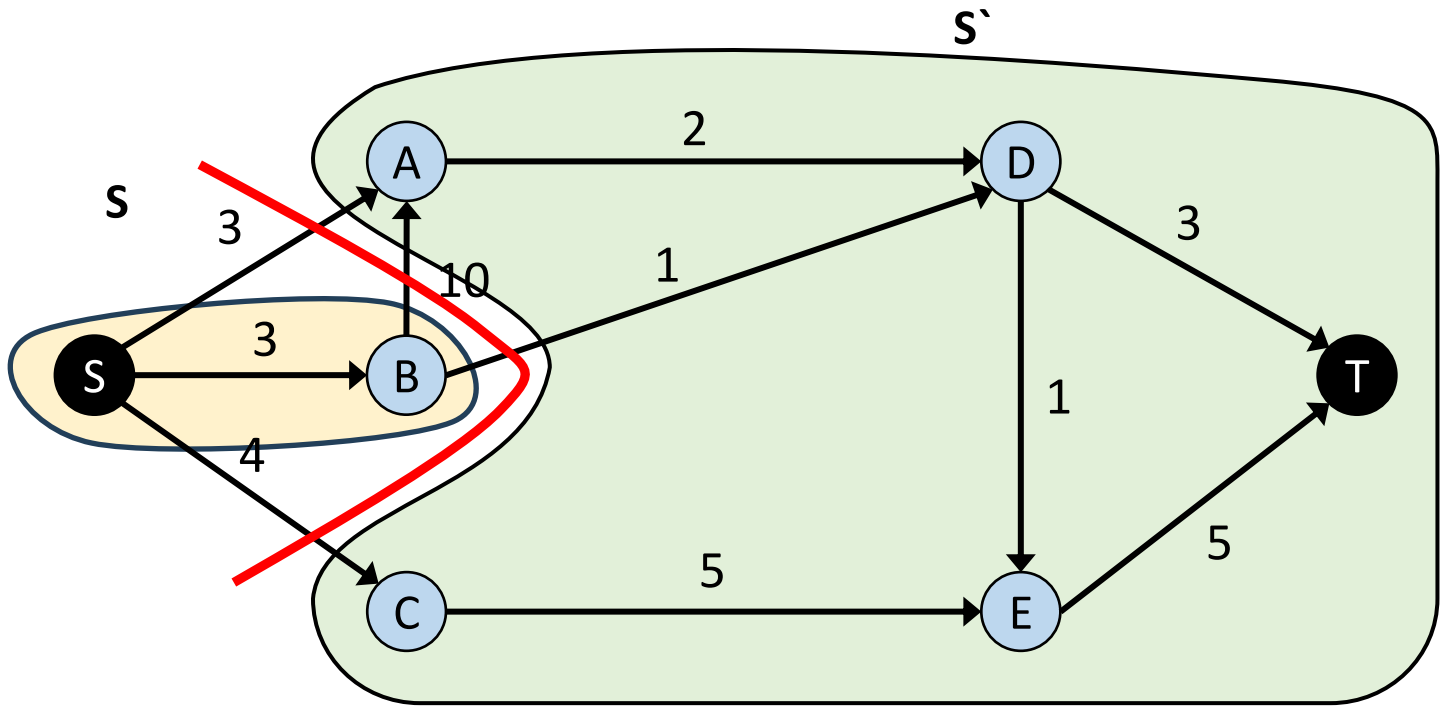


10 – 9

Max Flow – Min Cut (s-t cut)

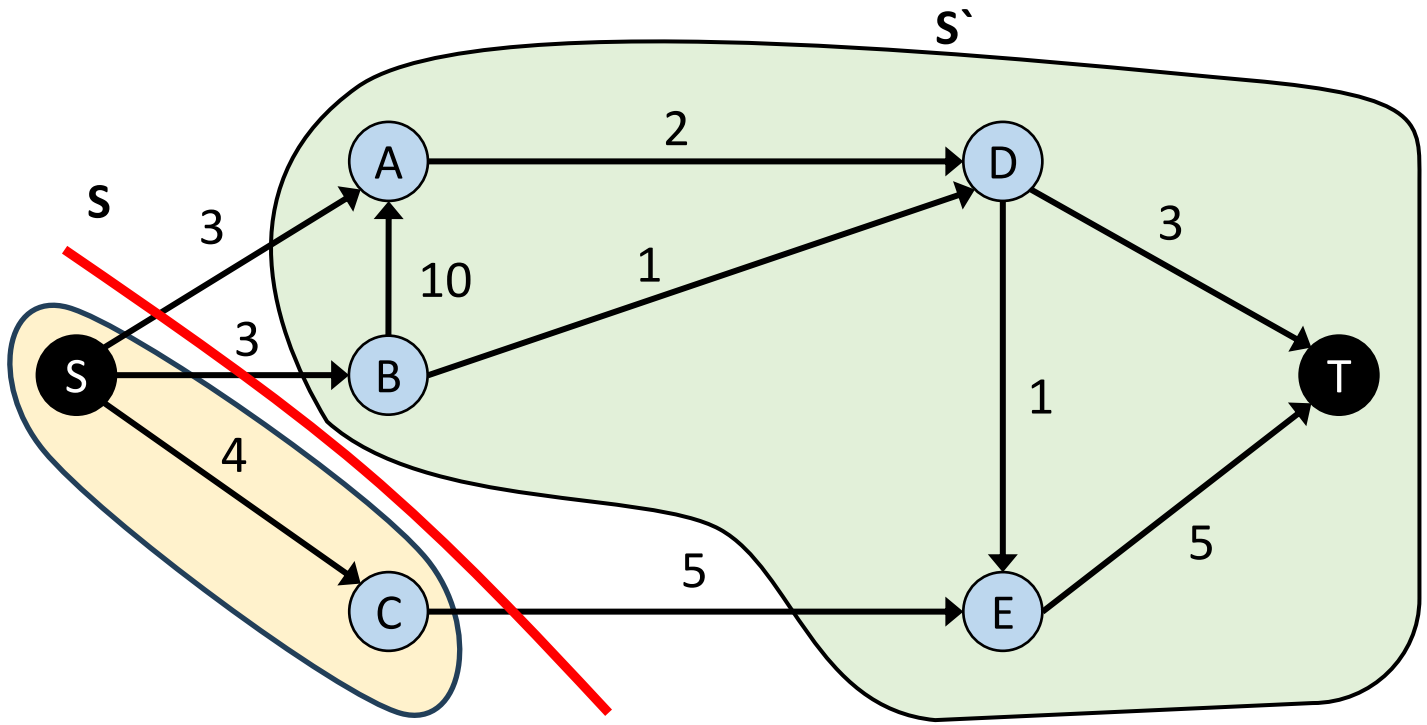


Max Flow – Min Cut (s-t cut)



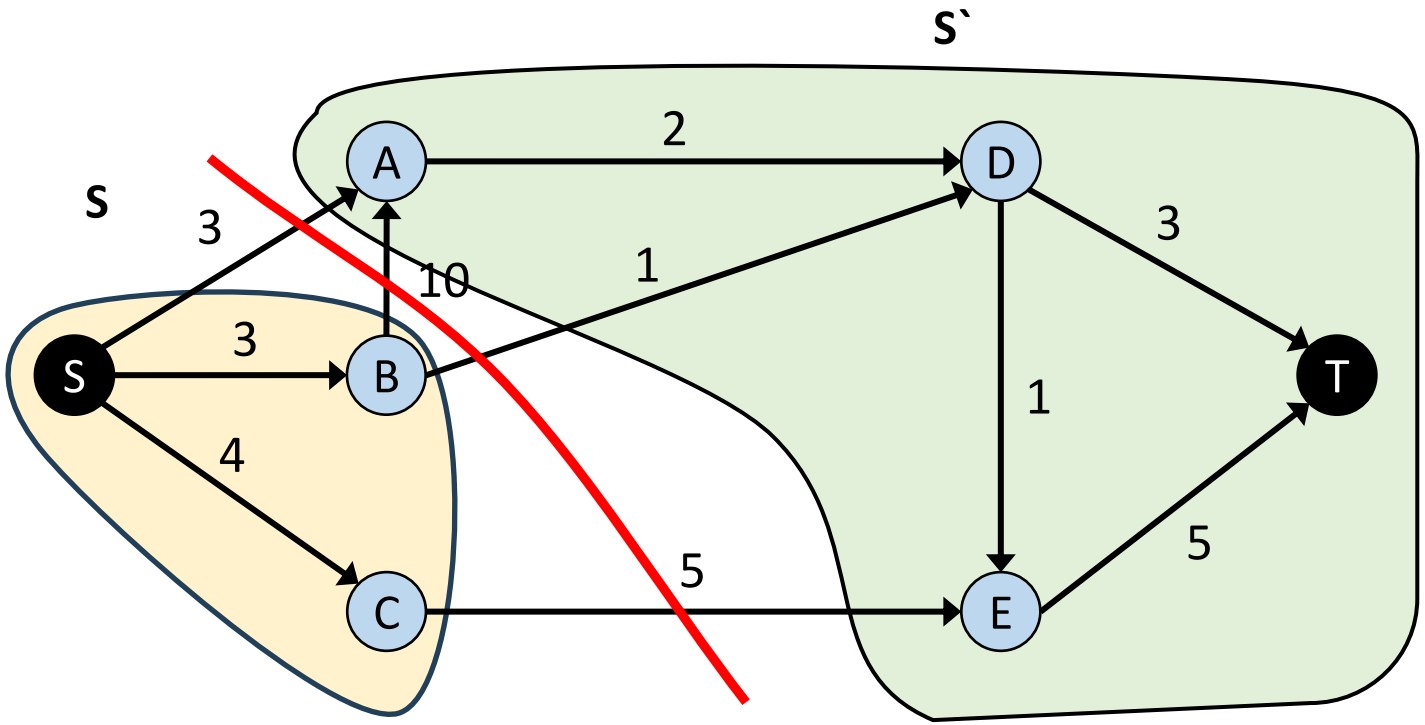
10 – 9 – 18

Max Flow – Min Cut (s-t cut)



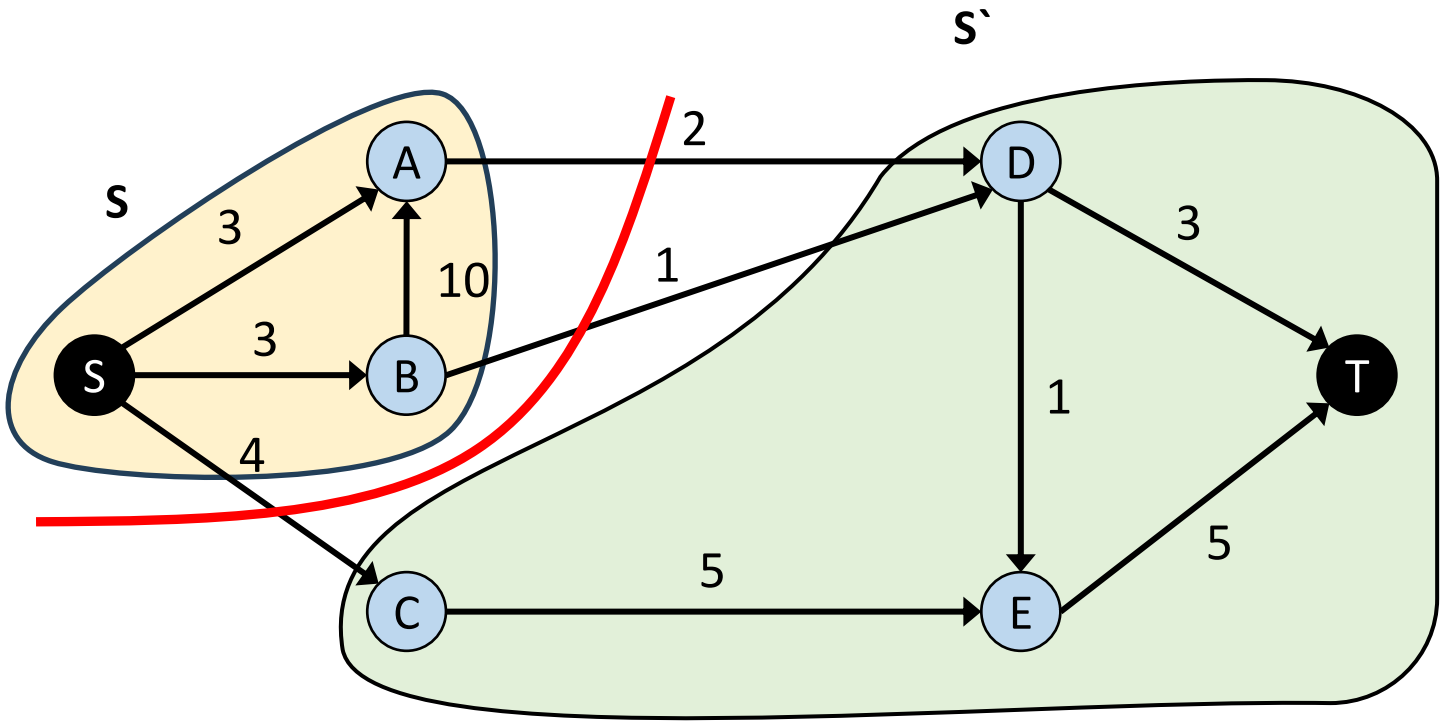
10 – 9 – 18 – 11

Max Flow – Min Cut (s-t cut)



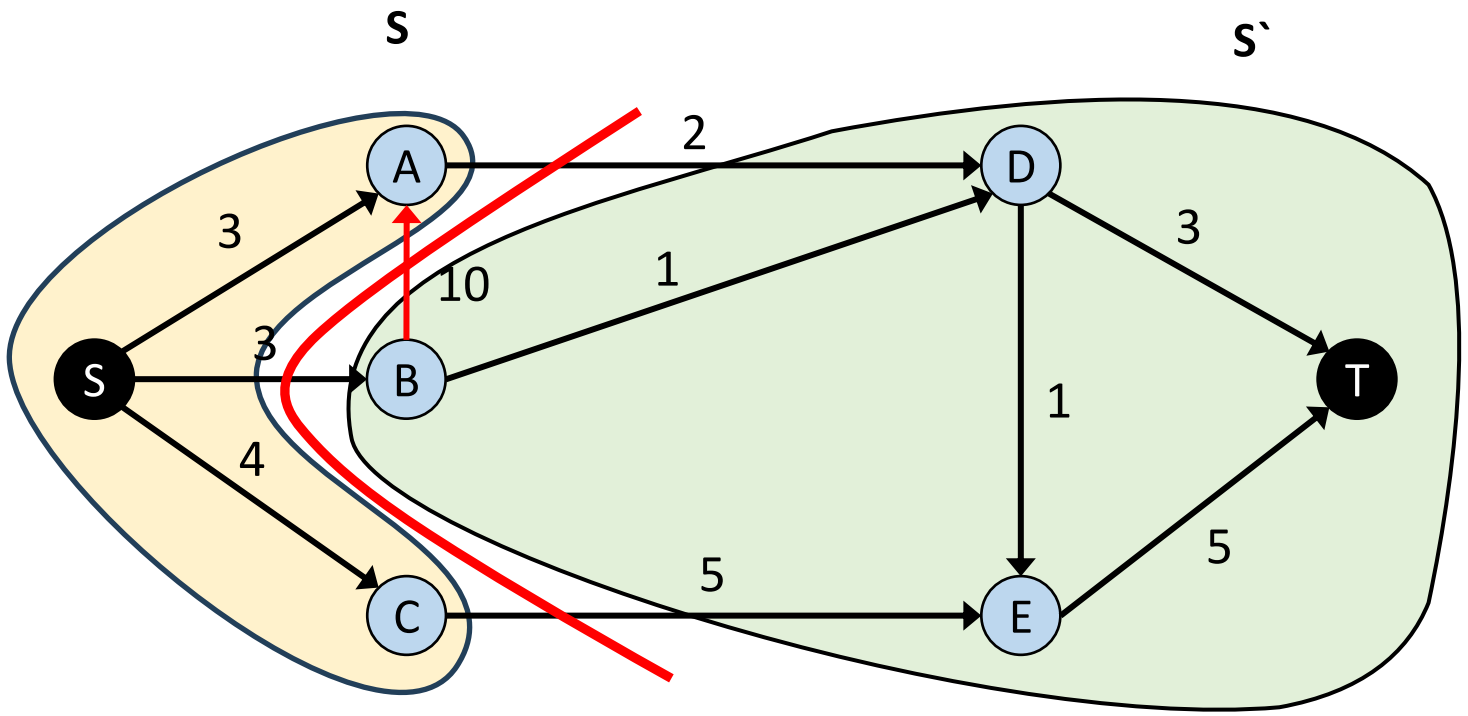
10 – 9 – 18 – 11 – 19

Max Flow – Min Cut (s-t cut)



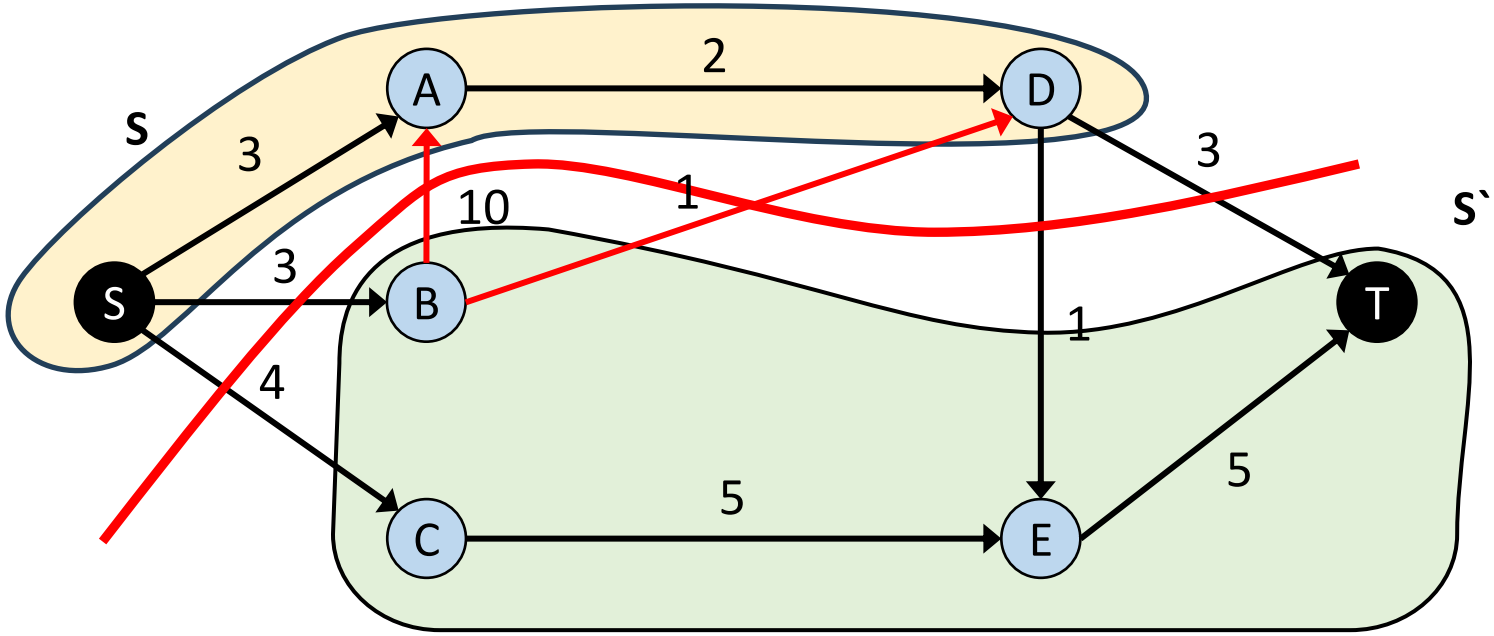
10 – 9 – 18 – 11 – 19 – 7

Max Flow – Min Cut (s-t cut)



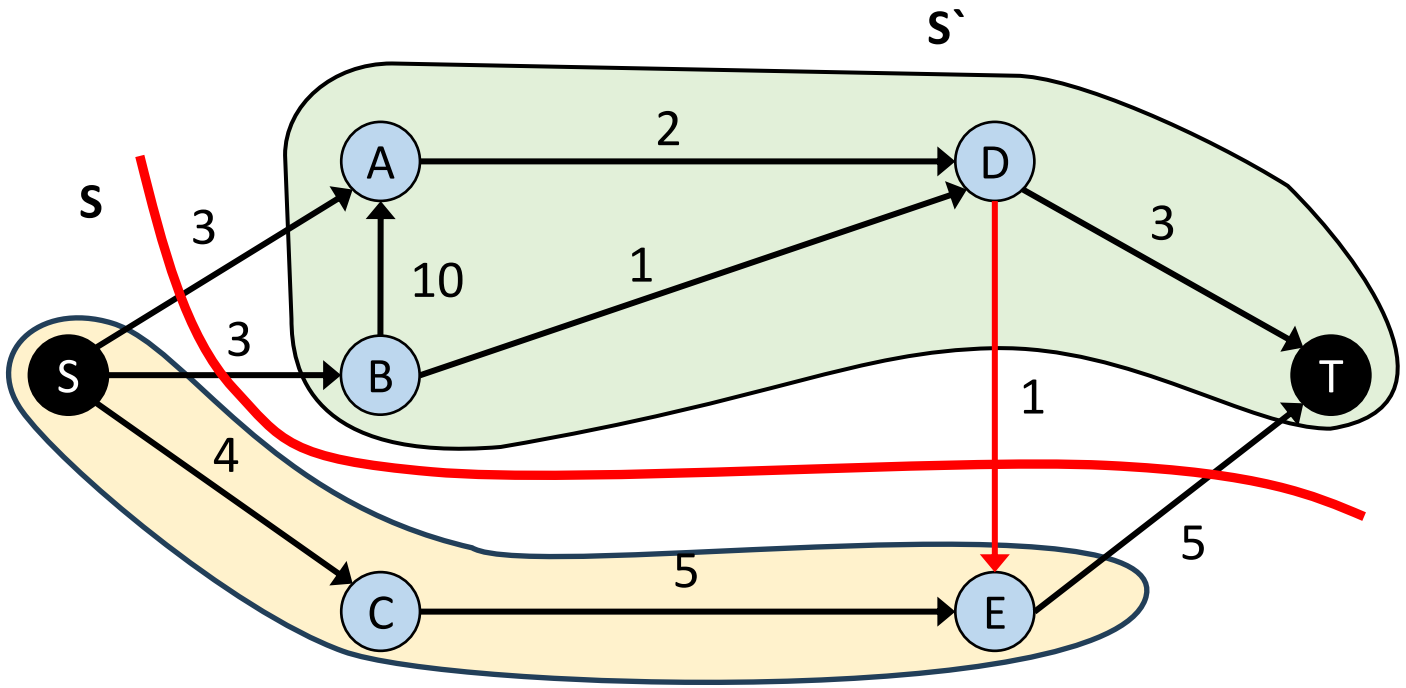
10 – 9 – 18 – 11 – 19 – 7 – 10

Max Flow – Min Cut (s-t cut)



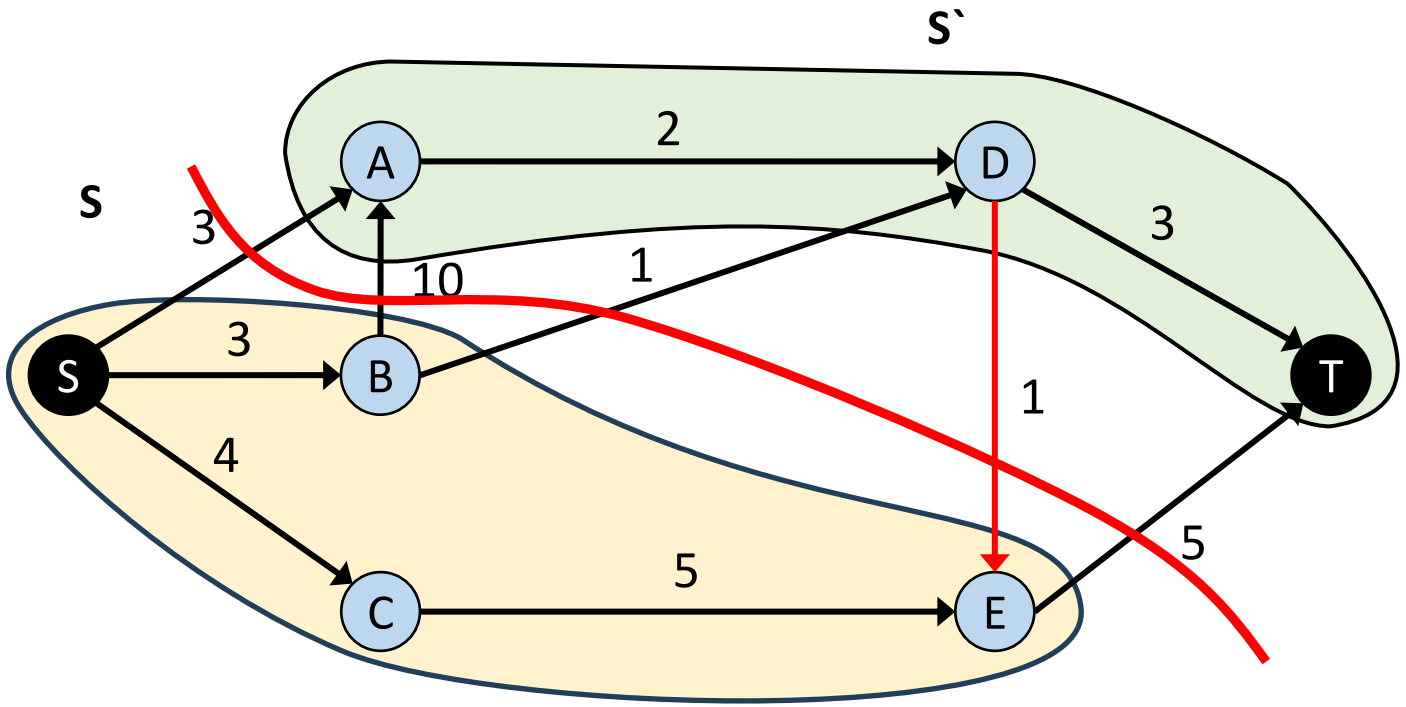
10 – 9 – 18 – 11 – 19 – 7 – 10 – 11

Max Flow – Min Cut (s-t cut)



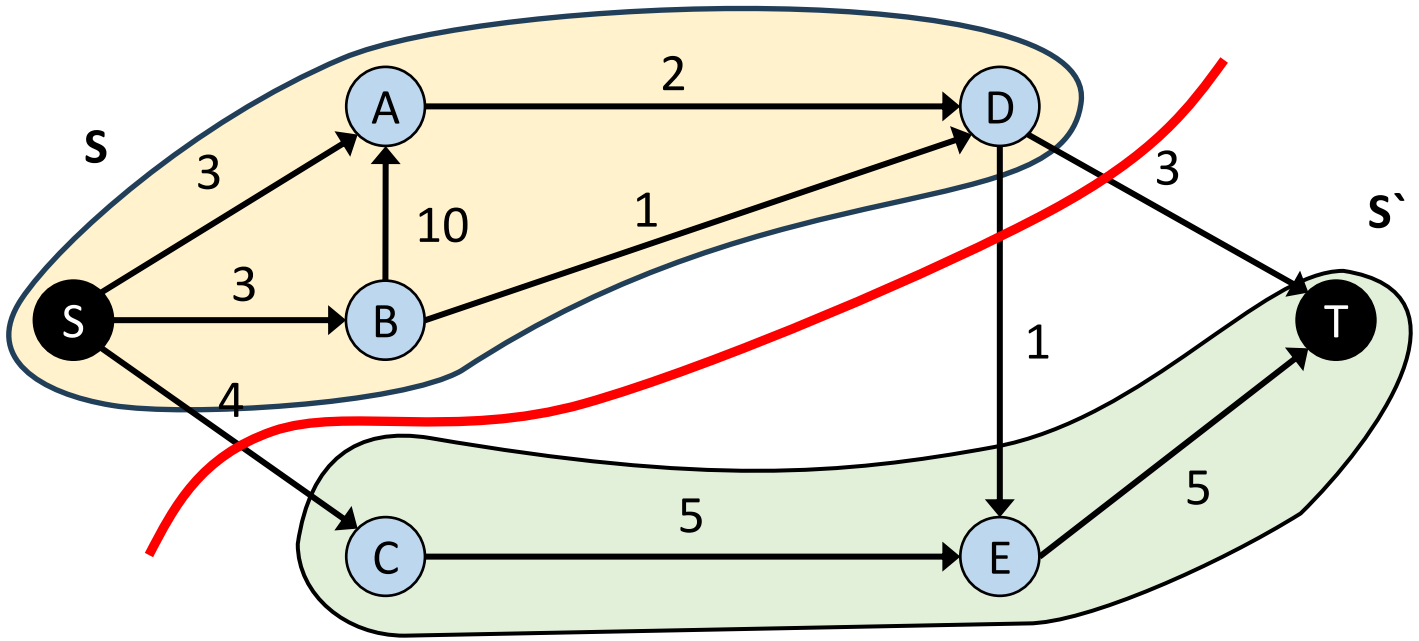
10 – 9 – 18 – 11 – 19 – 7 – 10 – 11 -11

Max Flow – Min Cut (s-t cut)



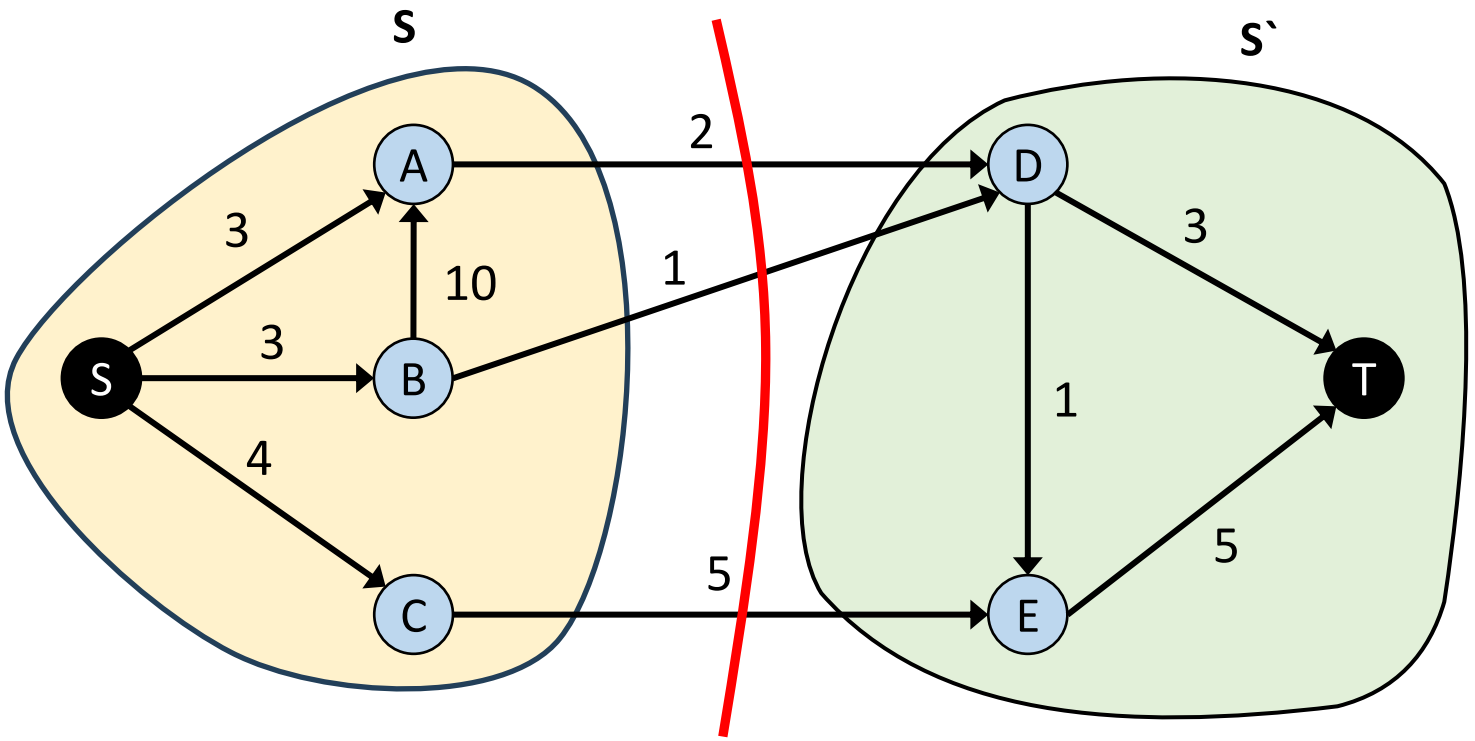
10 – 9 – 18 – 11 – 19 – 7 – 10 – 11 – 11 – 19

Max Flow – Min Cut (s-t cut)



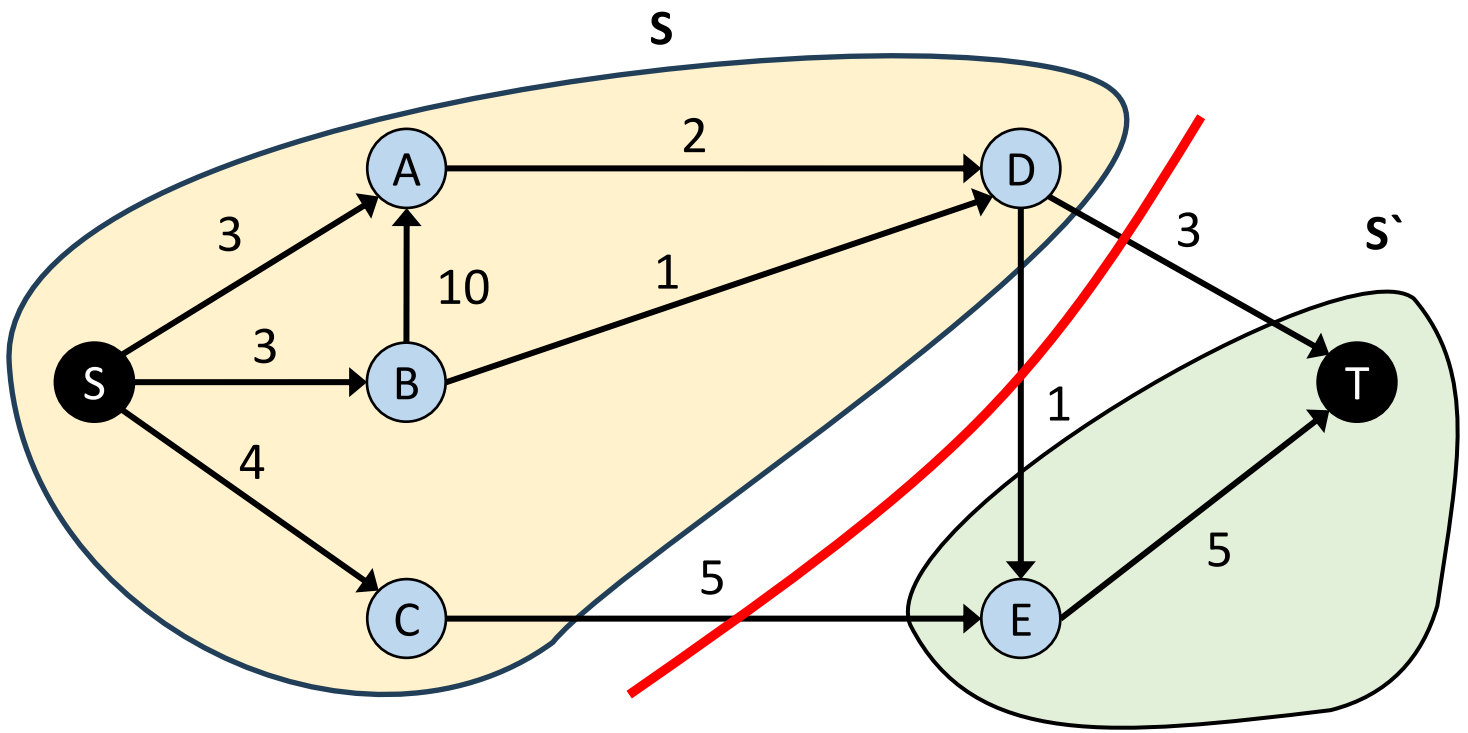
10 – 9 – 18 – 11 – 19 – 7 – 10 – 11 – 11 – 19 – 8

Max Flow – Min Cut (s-t cut)



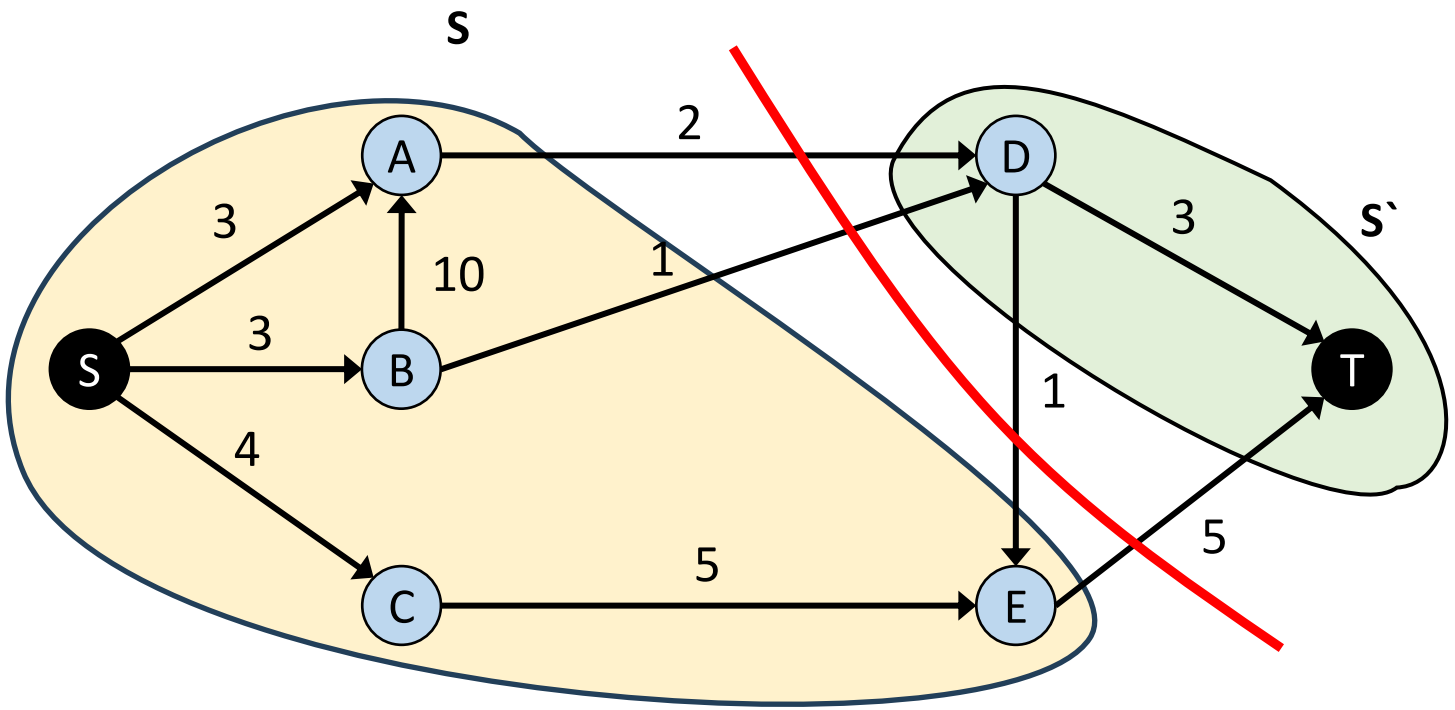
10 – 9 – 18 – 11 – 19 – 7 – 10 – 11 – 11 – 19 – 8 – 8

Max Flow – Min Cut (s-t cut)



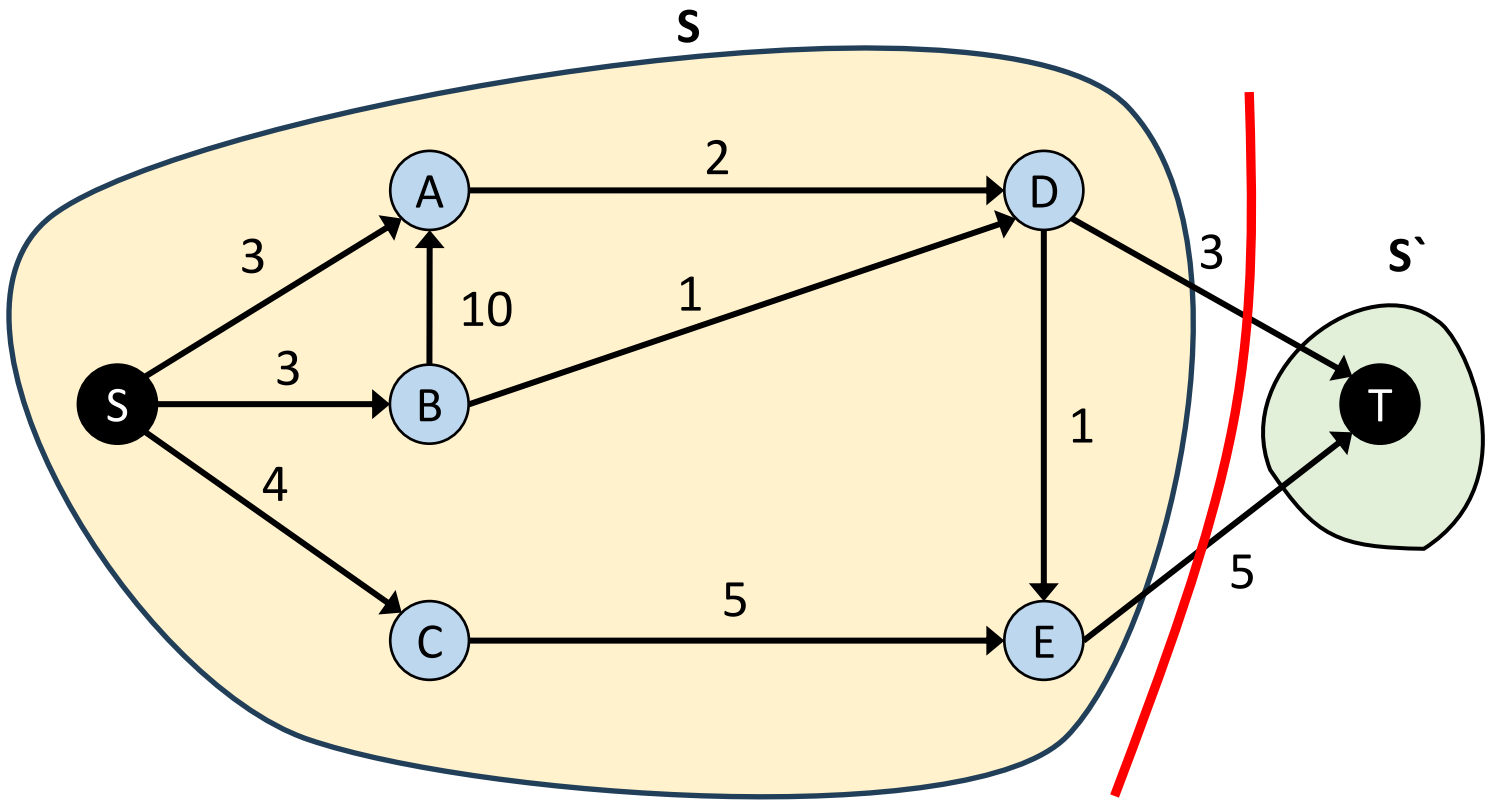
10 – 9 – 18 – 11 – 19 – 7 – 10 – 11 – 11 – 19 – 8 – 9

Max Flow – Min Cut (s-t cut)



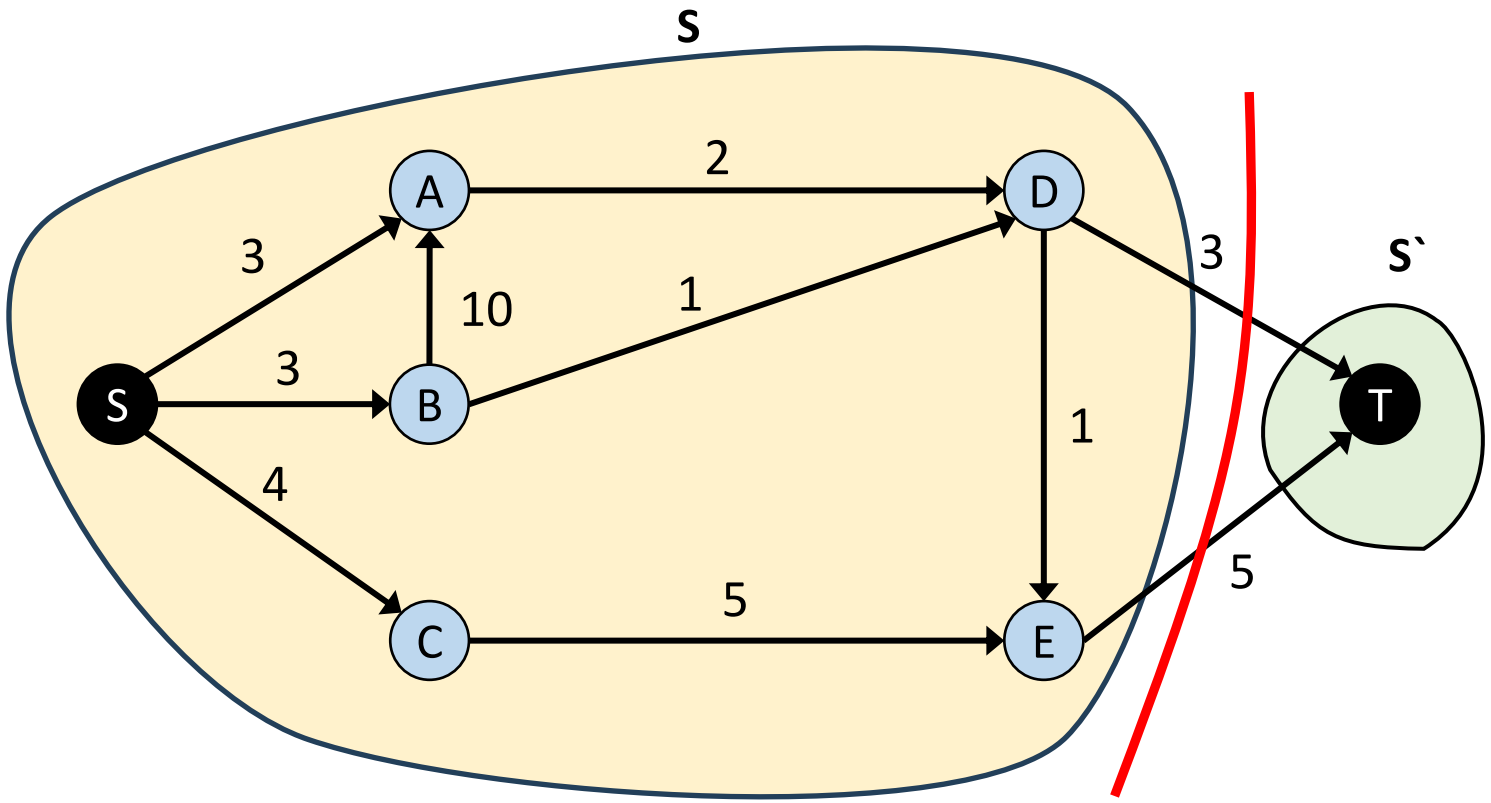
10 – 9 – 18 – 11 – 19 – 7 – 10 – 11 – 11 – 19 – 8 – 9

Max Flow – Min Cut (s-t cut)



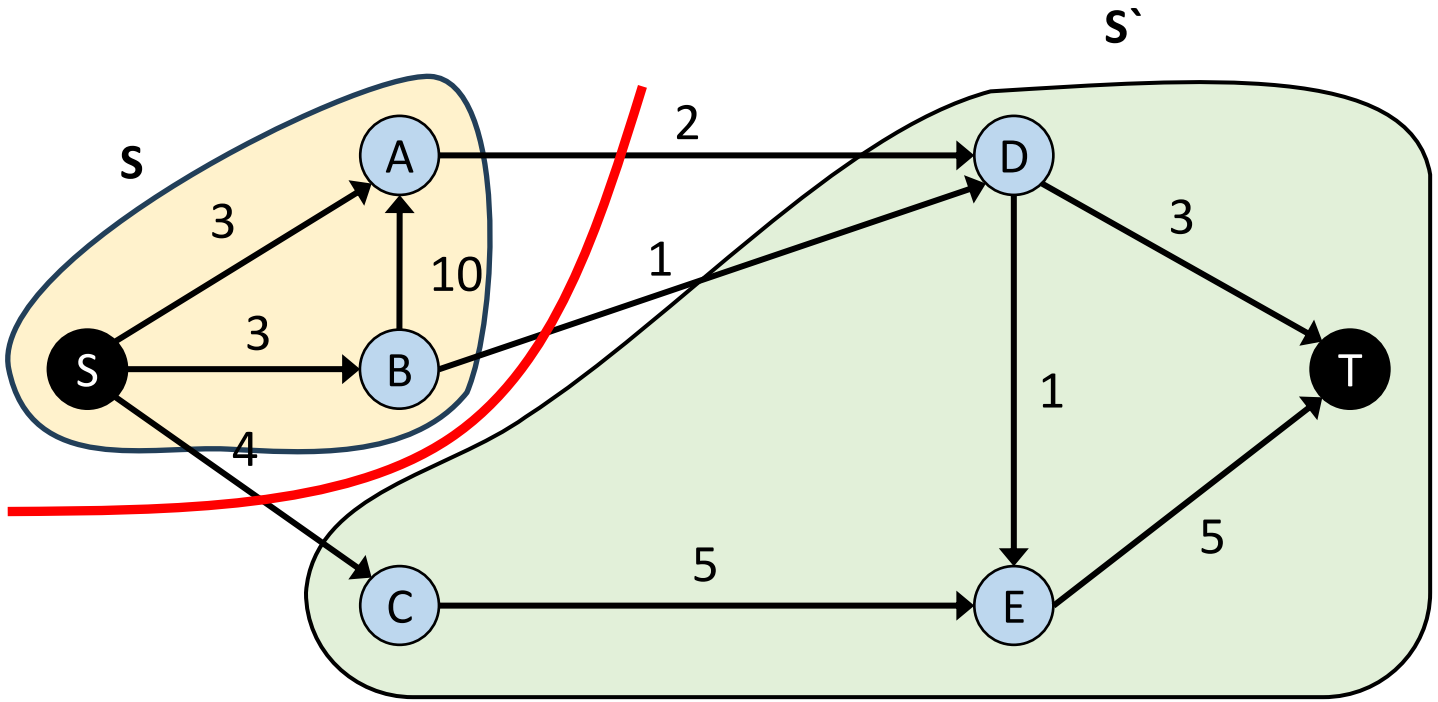
10 – 9 – 18 – 11 – 19 – 7 – 10 – 11 – 11 – 19 – 8 – 9 – 8

Max Flow – Min Cut (s-t cut)



10 – 9 – 18 – 11 – 19 – **7** – 10 – 11 – 11 – 19 – 8 – 9 – 8

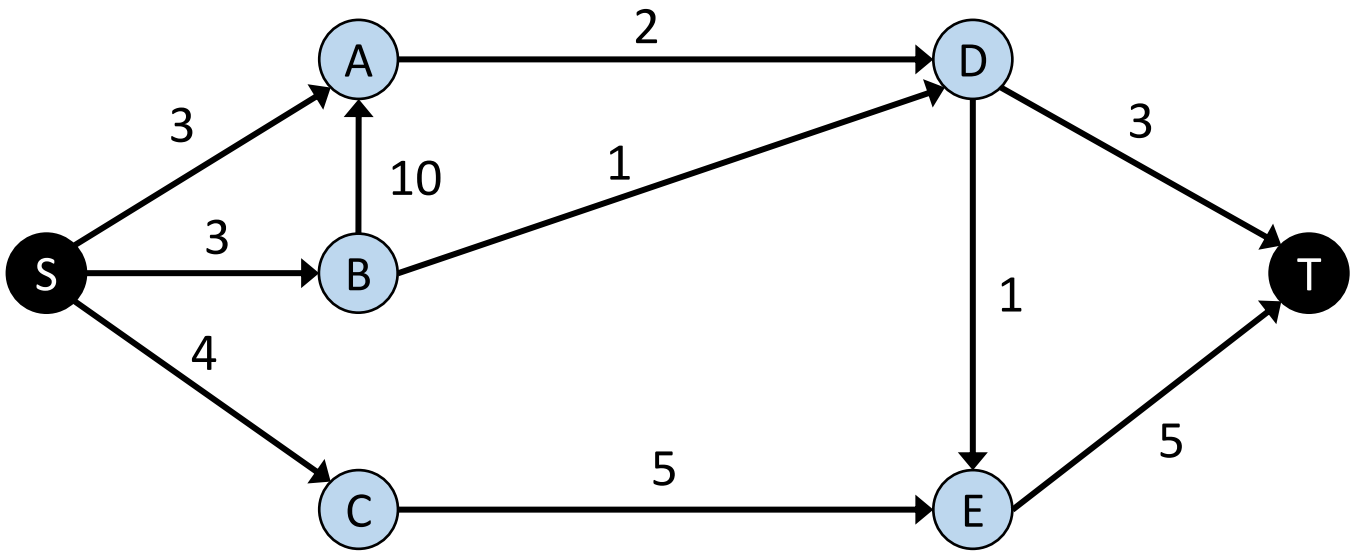
Max Flow – Min Cut (s-t cut)



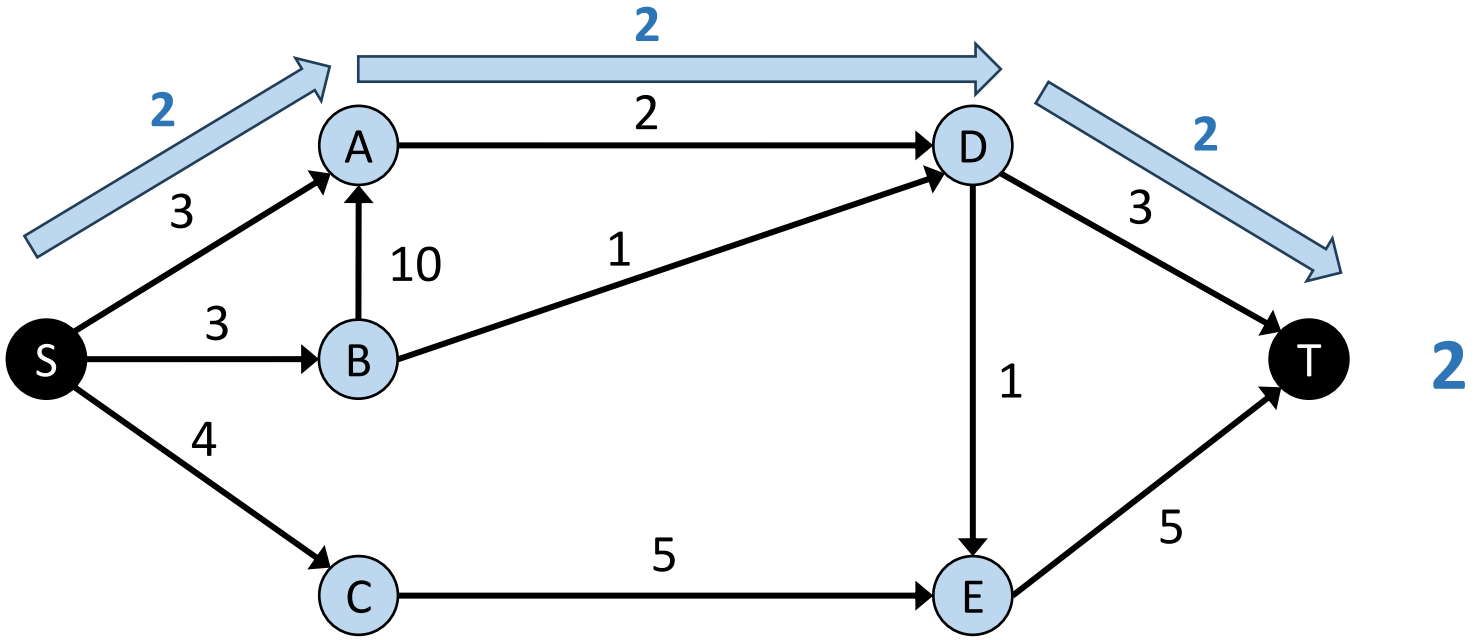
Tightest upper bound will come from a $s - t$ cut of minimum capacity

10 – 9 – 18 – 11 – 19 – **7** – 10 – 11 – 11 – 19 – 8 – 9 – 8

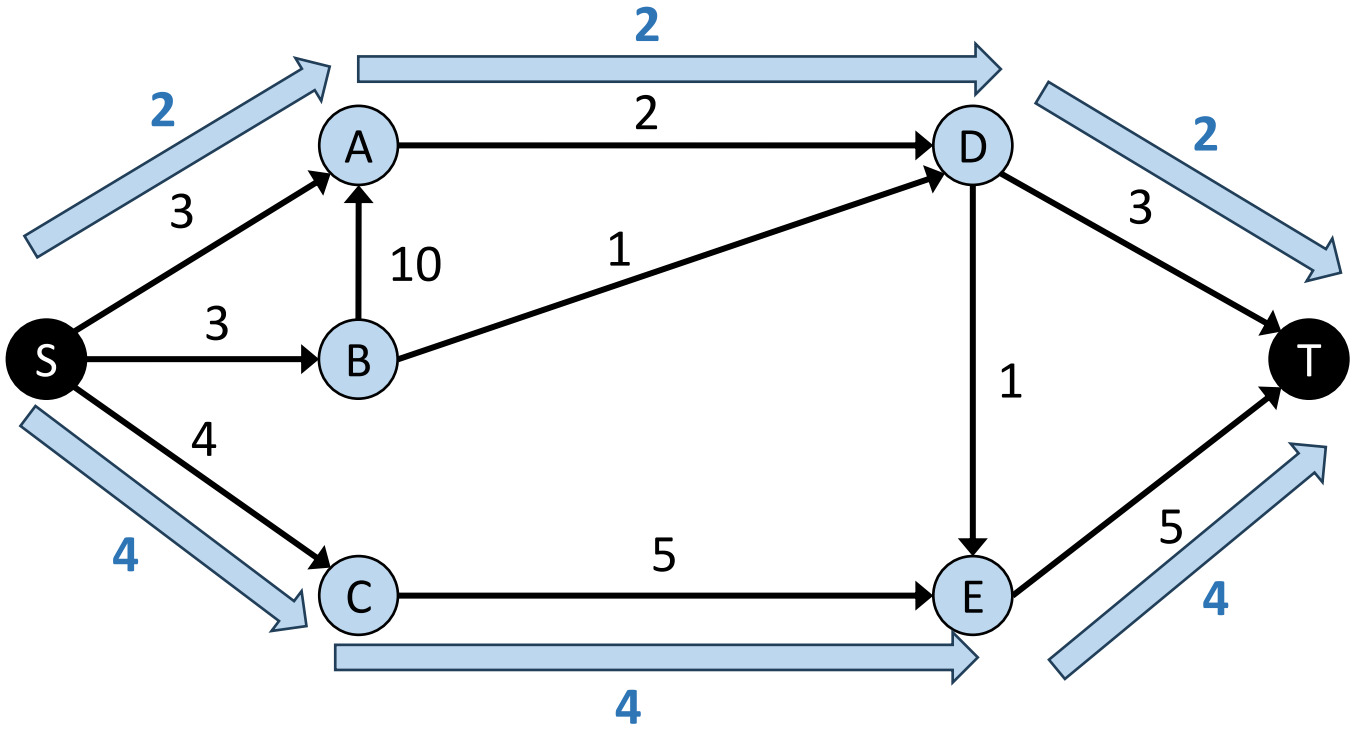
Max Flow – Upper limit



Max Flow – Upper limit

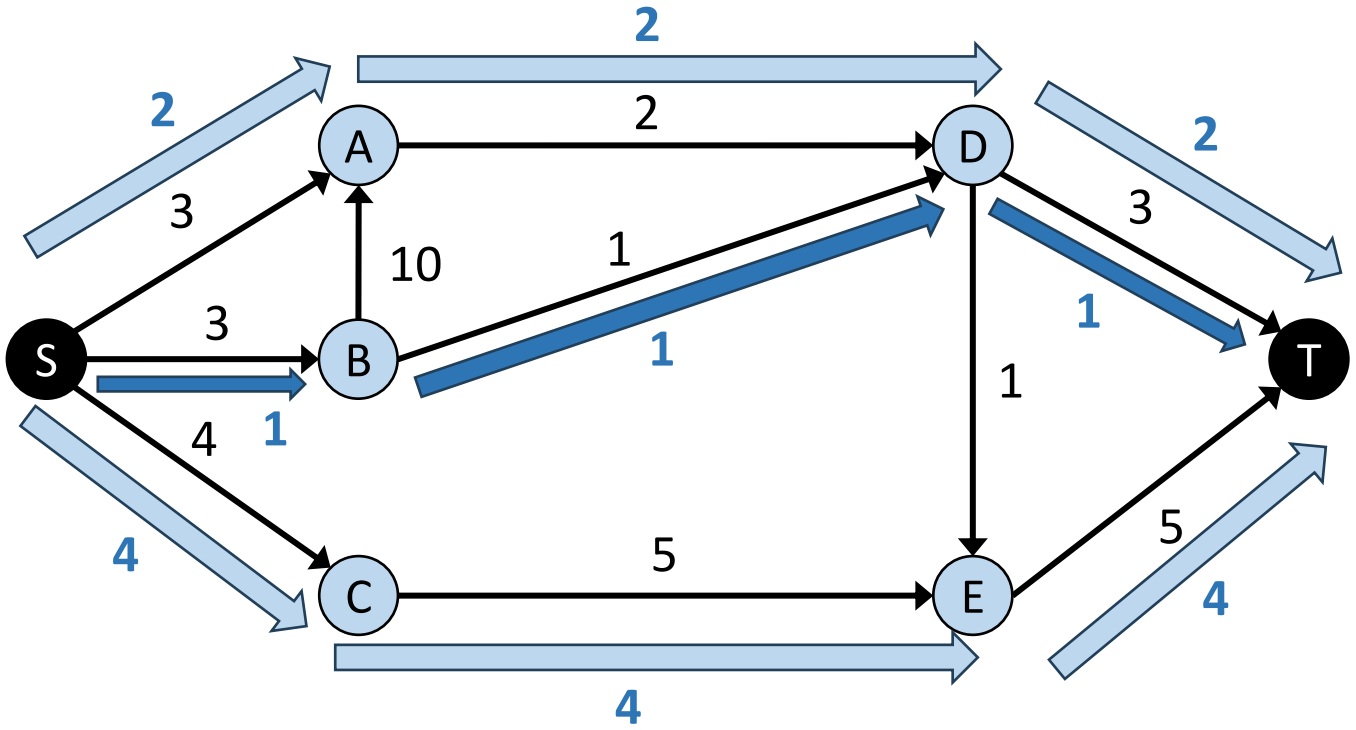


Max Flow – Upper limit



$$size(f) = f^{out}(s) = f^{in}(t)$$

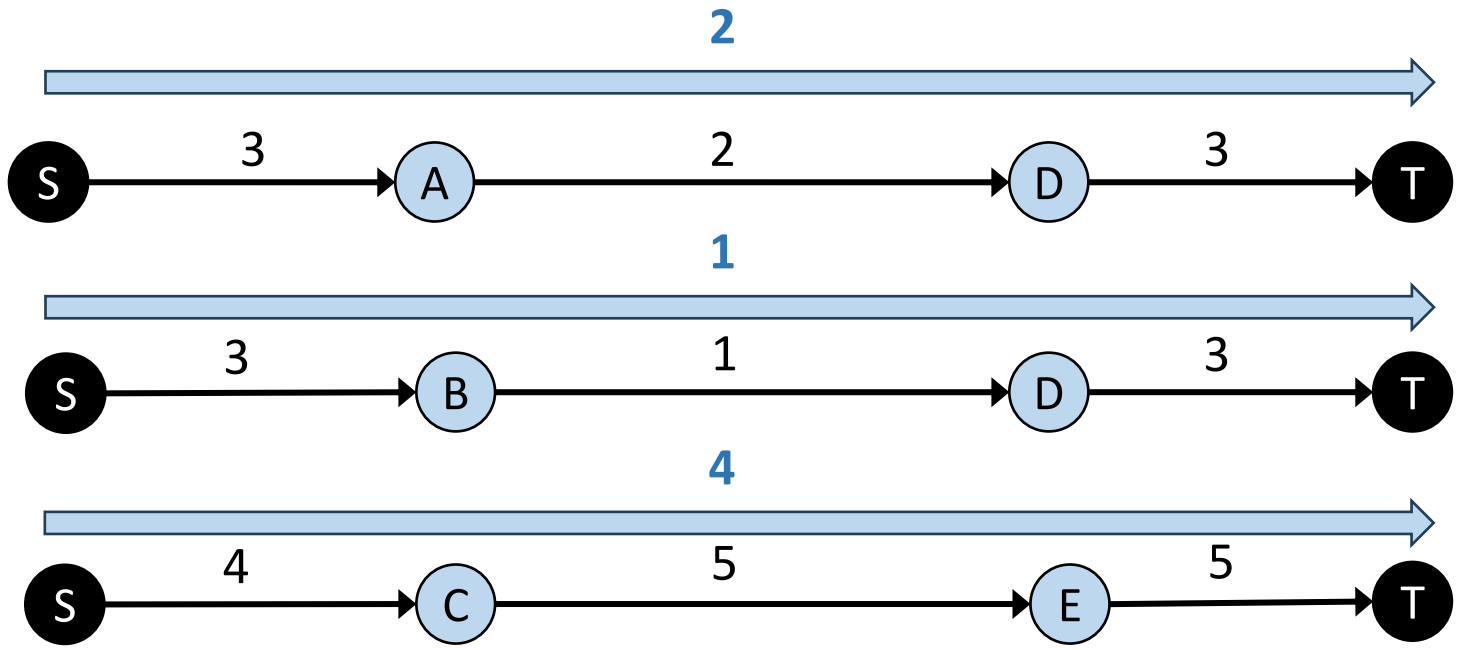
Max Flow – Upper limit



2 + 4 + 1 = 7

$$size(f) = f^{out}(s) = f^{in}(t)$$

Max Flow – Bottleneck



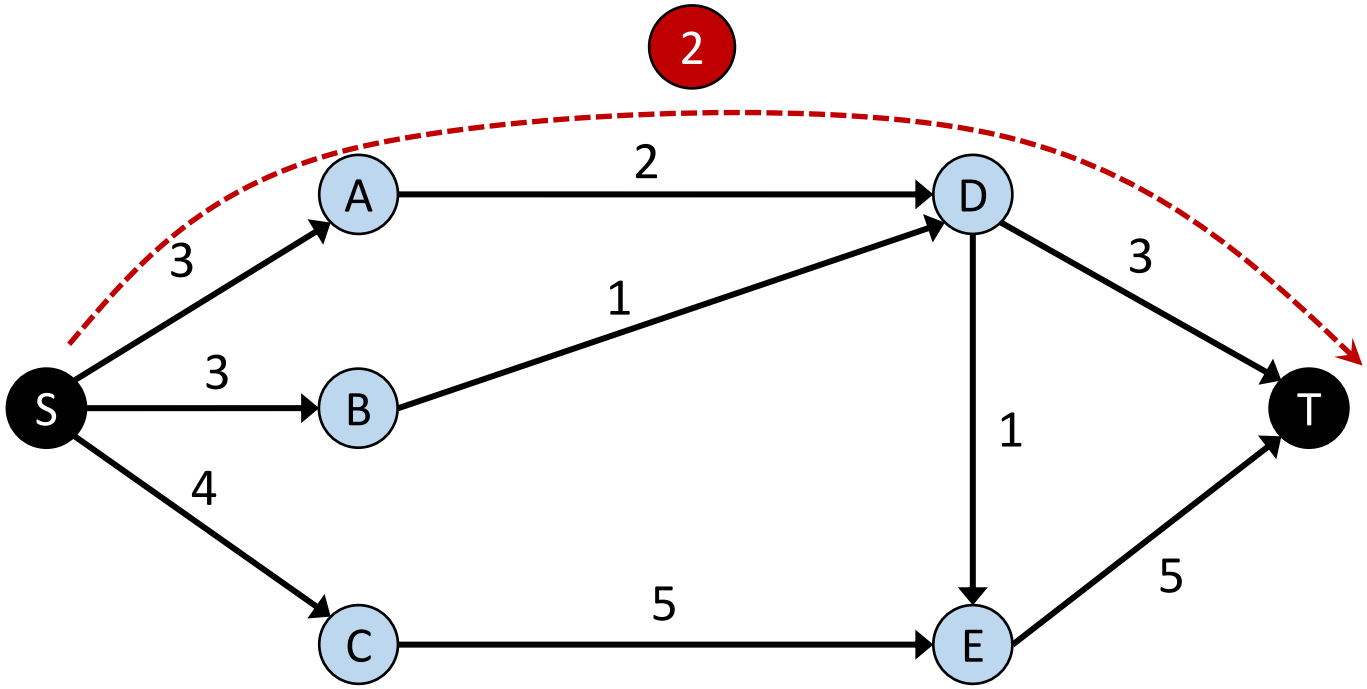
Bottleneck= 2

Bottleneck= 1

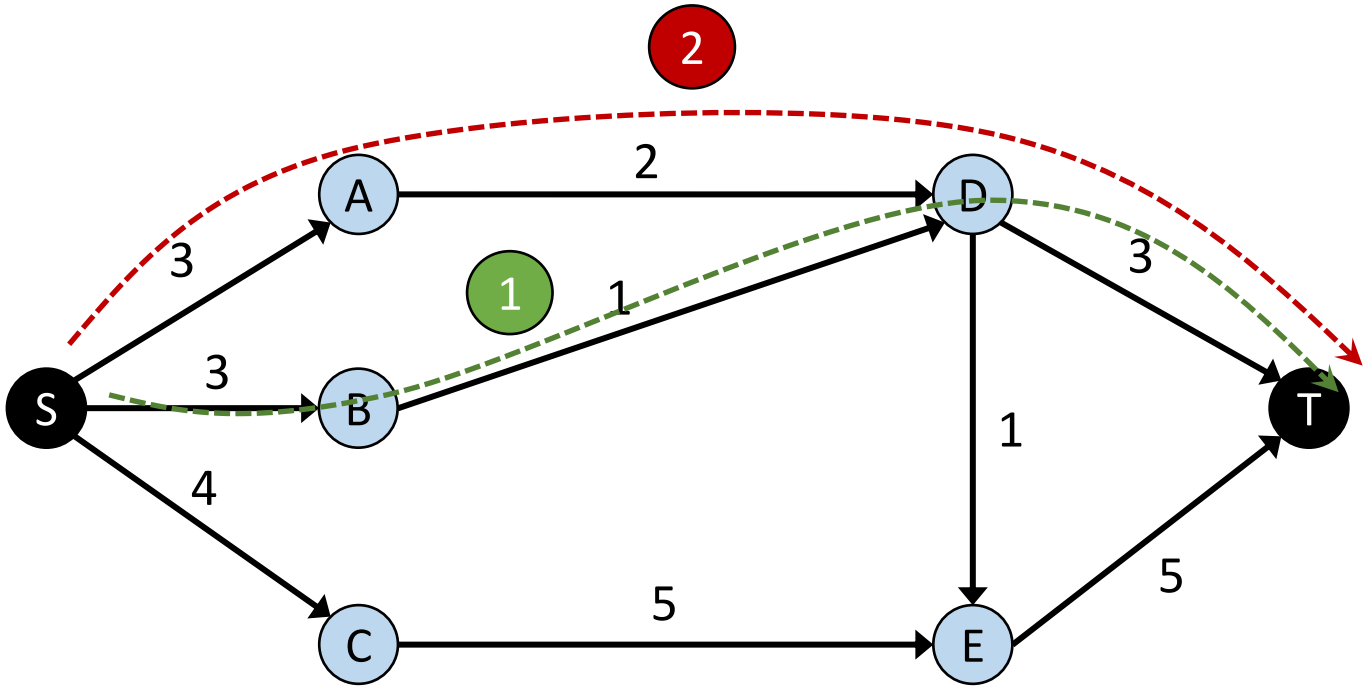
Bottleneck= 4

Bottleneck= minimum c_e in the path

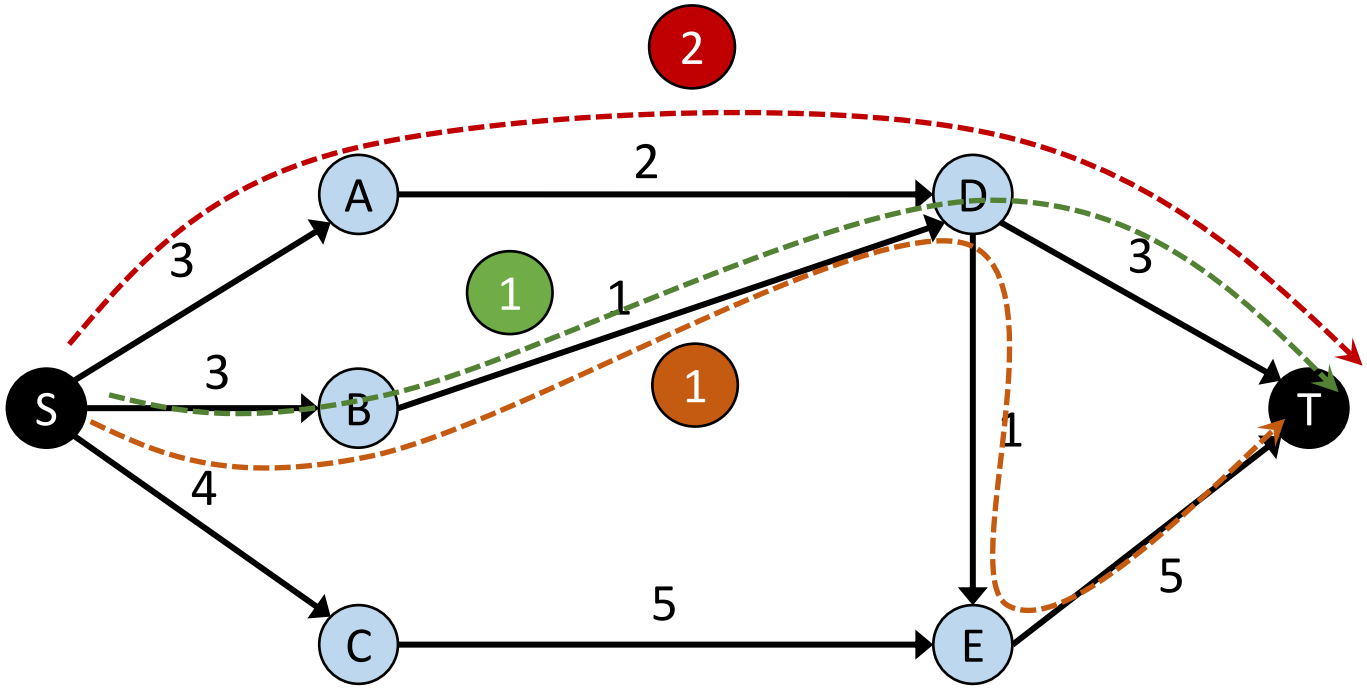
Max Flow – Algorithm (first try - greedy)



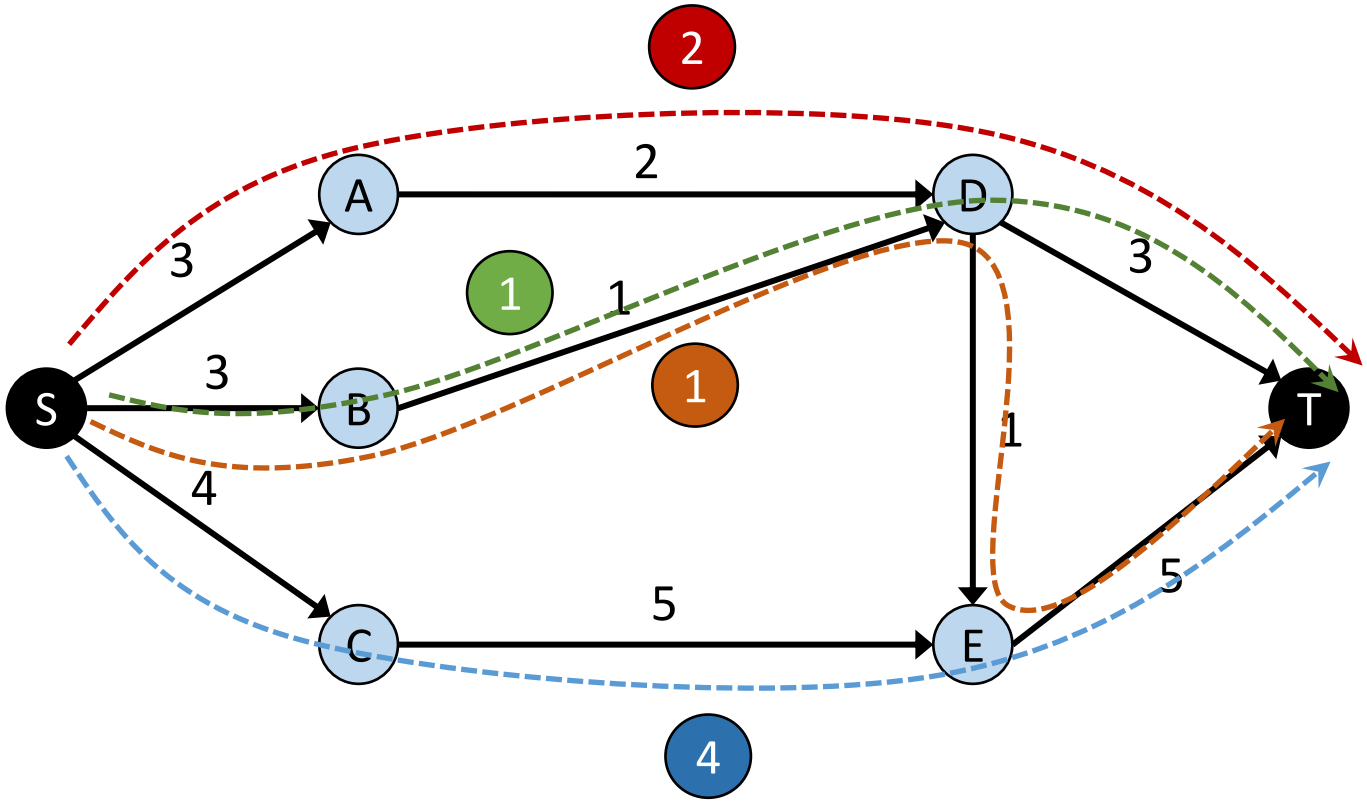
Max Flow – Algorithm (first try - greedy)



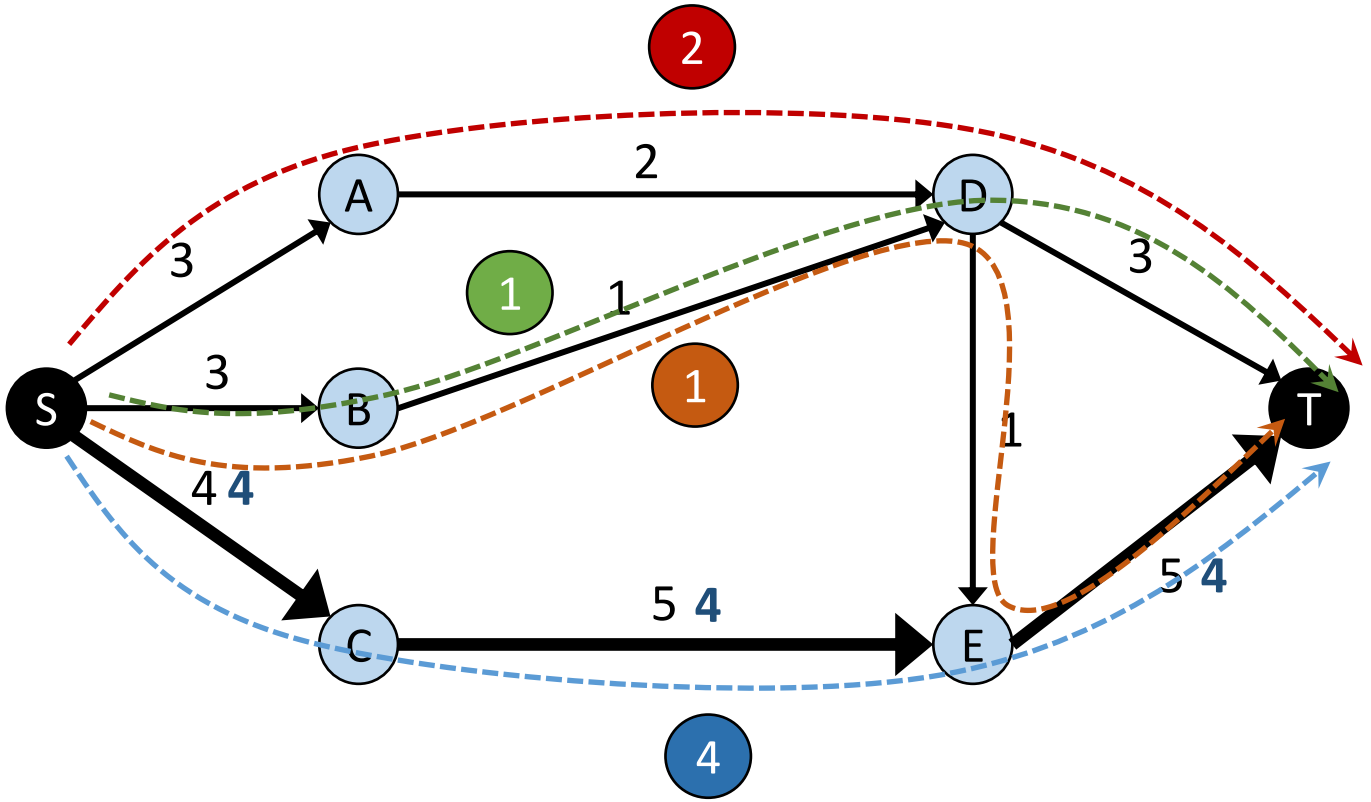
Max Flow – Algorithm (first try - greedy)



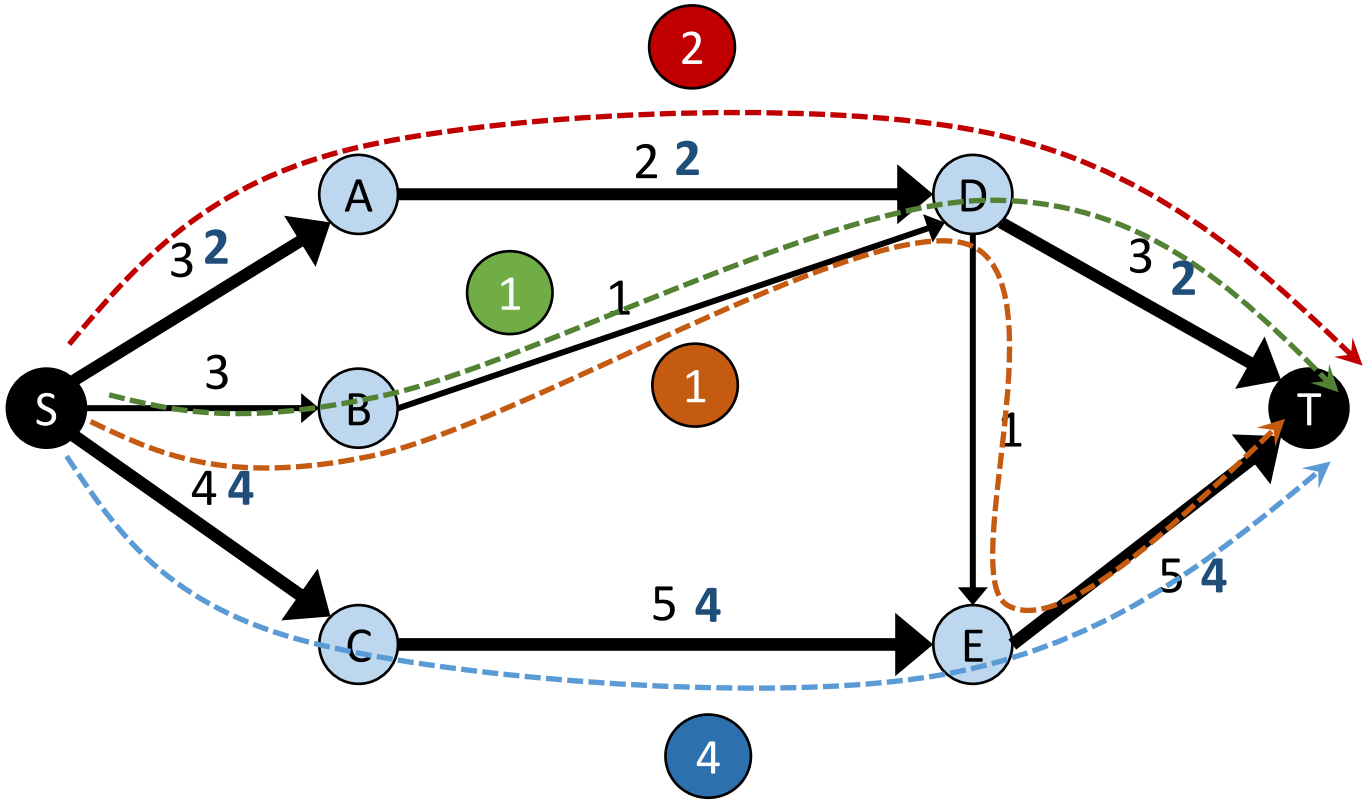
Max Flow – Algorithm (first try - greedy)



Max Flow – Algorithm (first try - greedy)

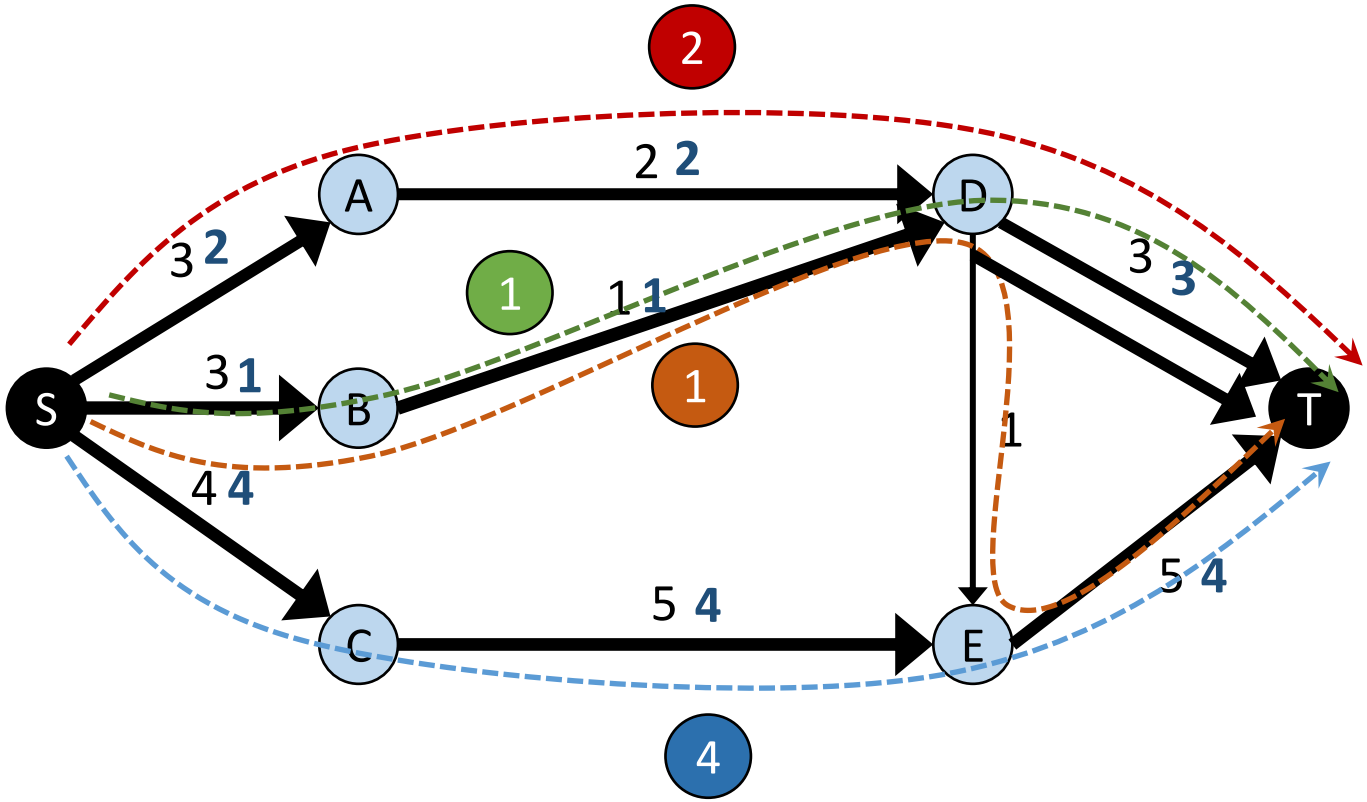


Max Flow – Algorithm (first try - greedy)



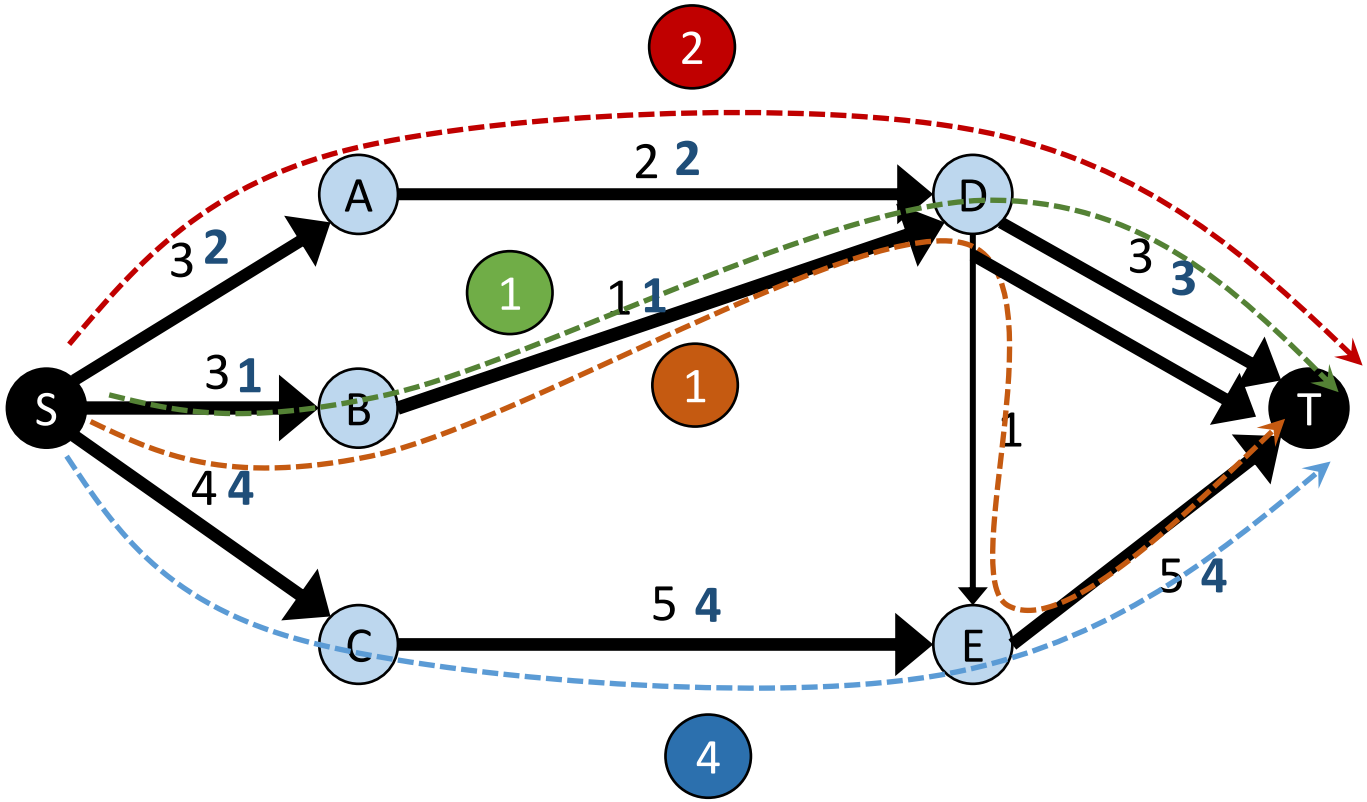
4 + 2 +

Max Flow – Algorithm (first try - greedy)



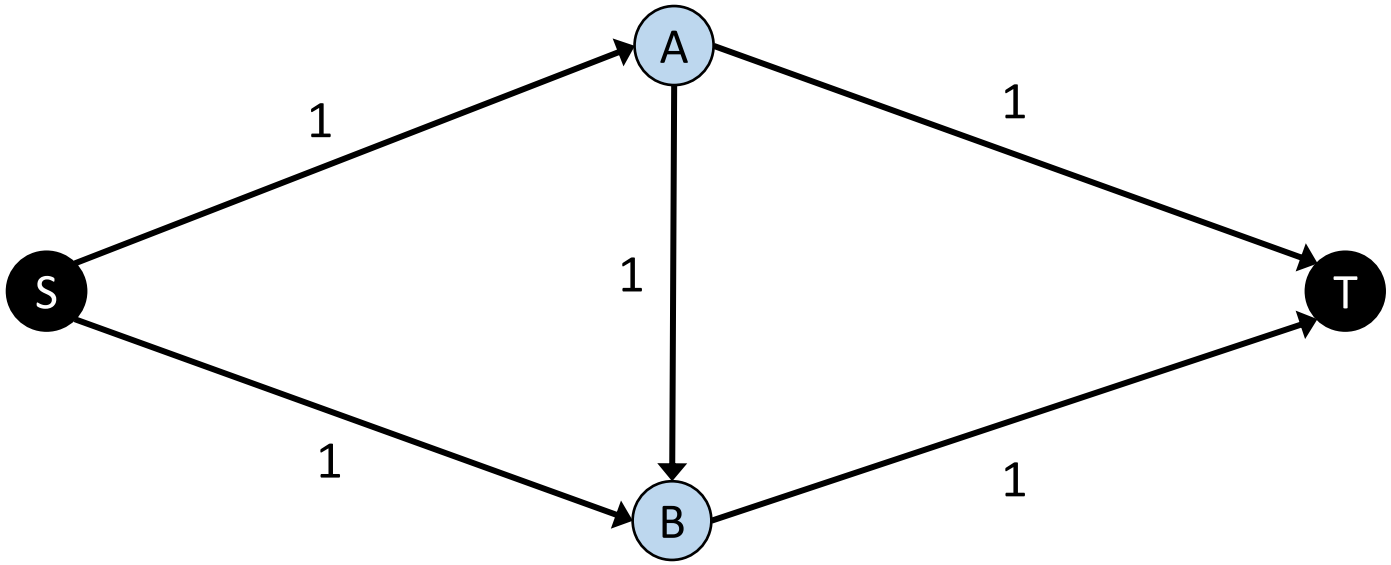
4 + 2 + 1

Max Flow – Algorithm (first try - greedy)

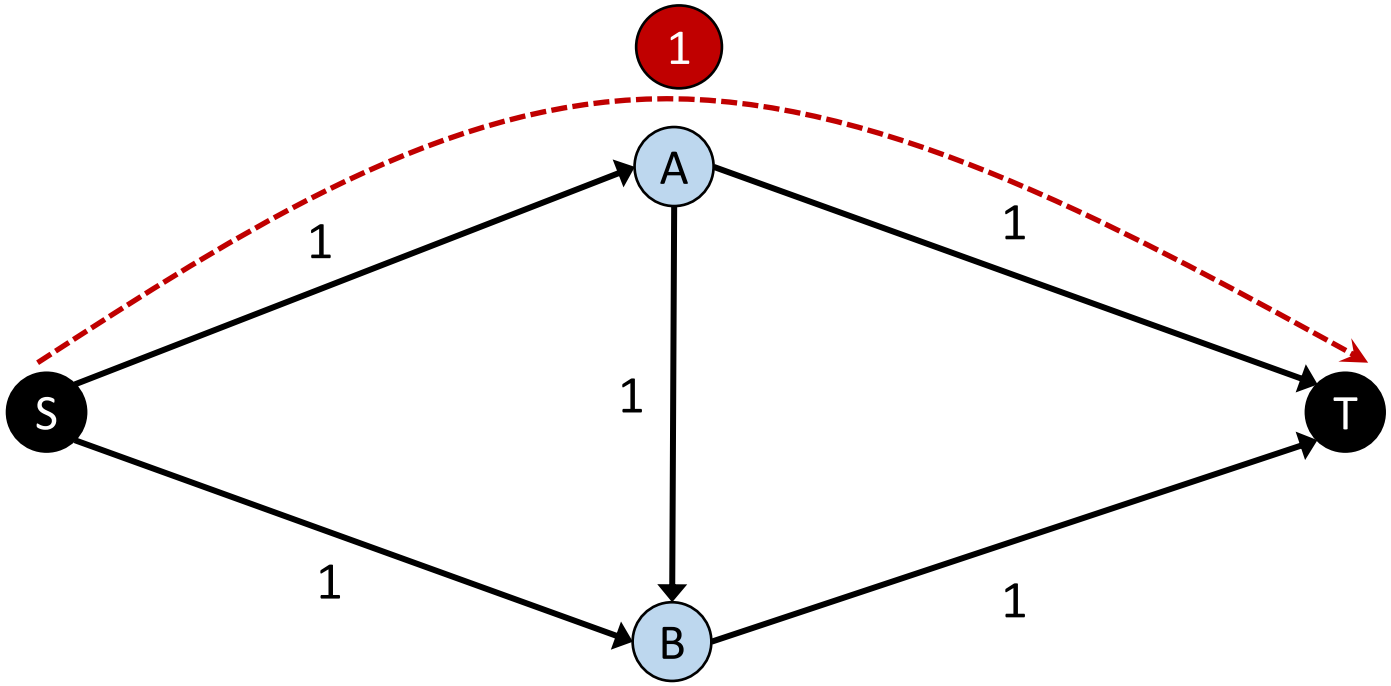


4 + 2 + 1 = 7

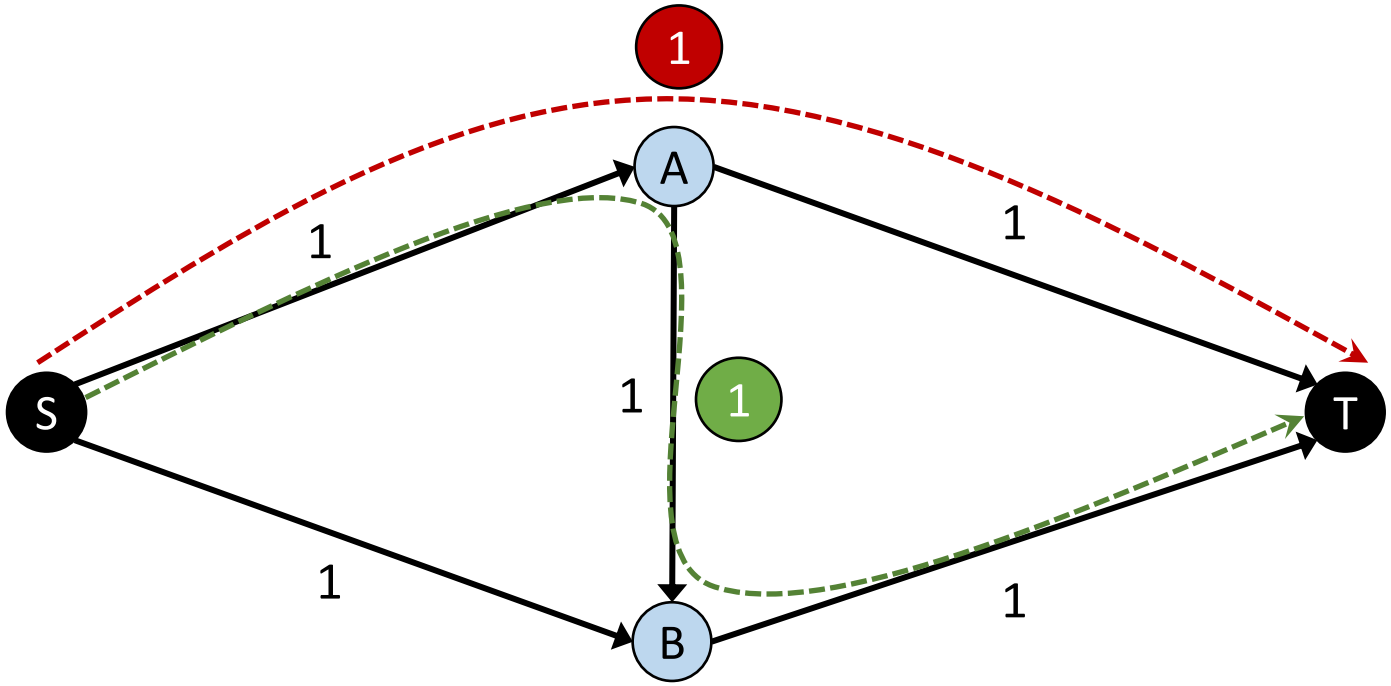
Max Flow – Problem with the Algorithm



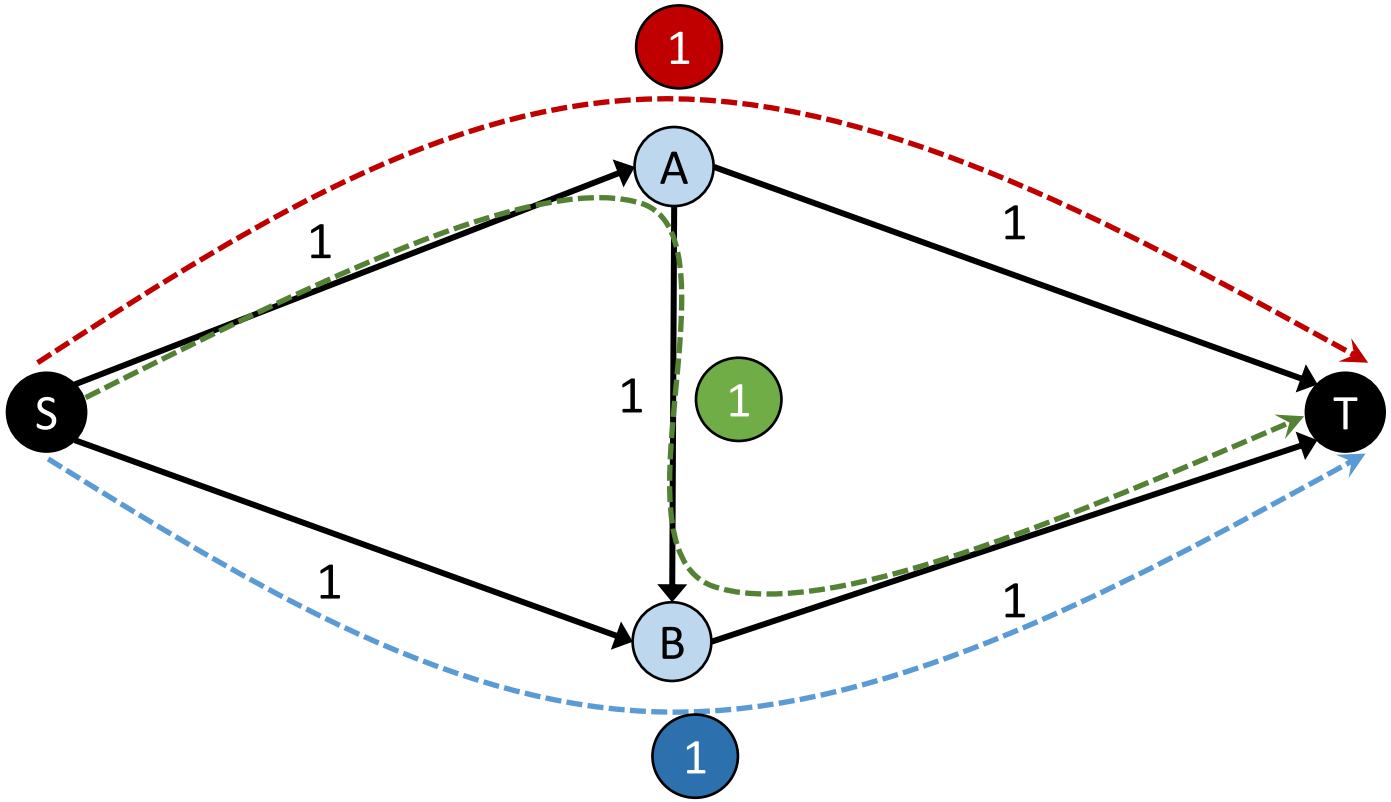
Max Flow – Problem with the Algorithm



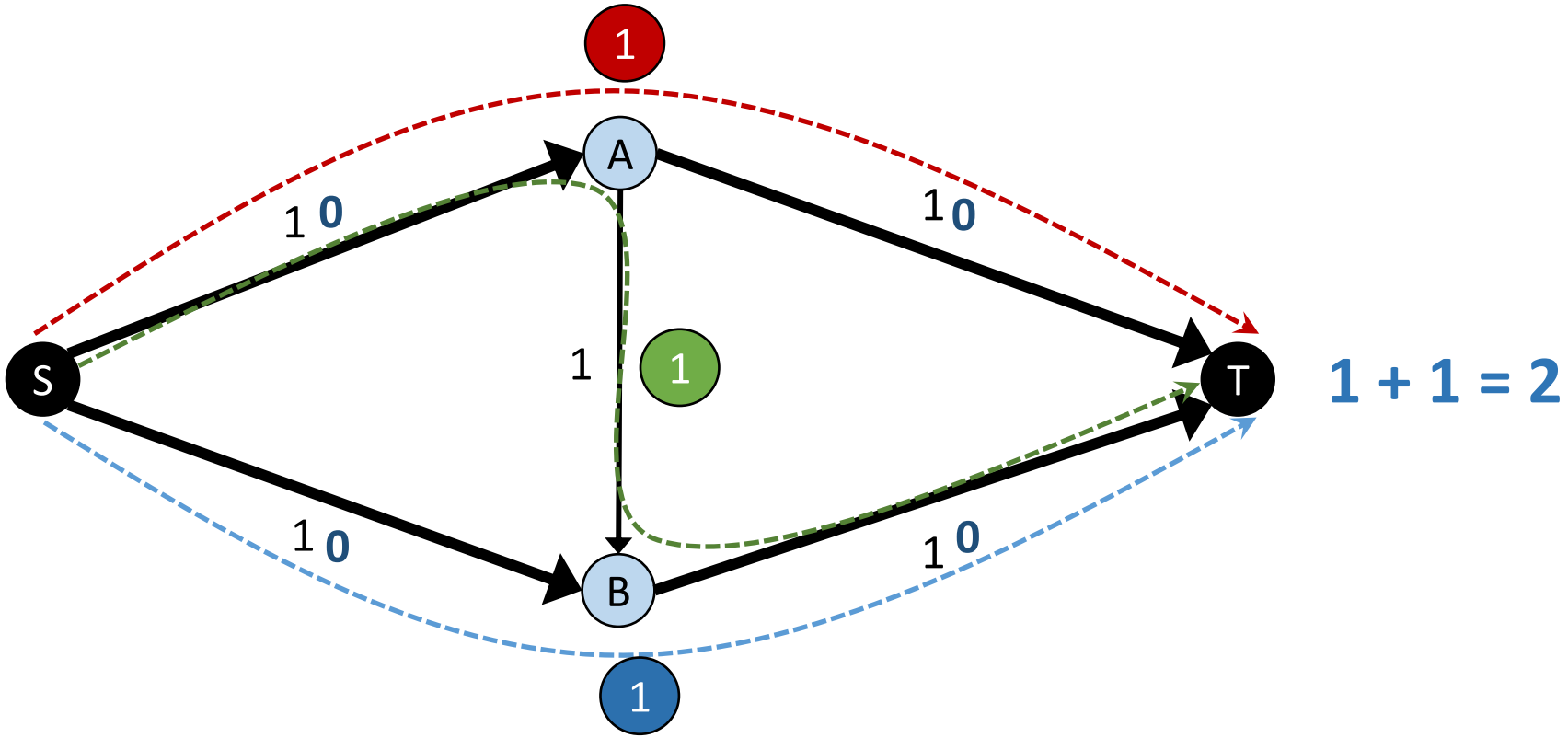
Max Flow – Problem with the Algorithm



Max Flow – Problem with the Algorithm

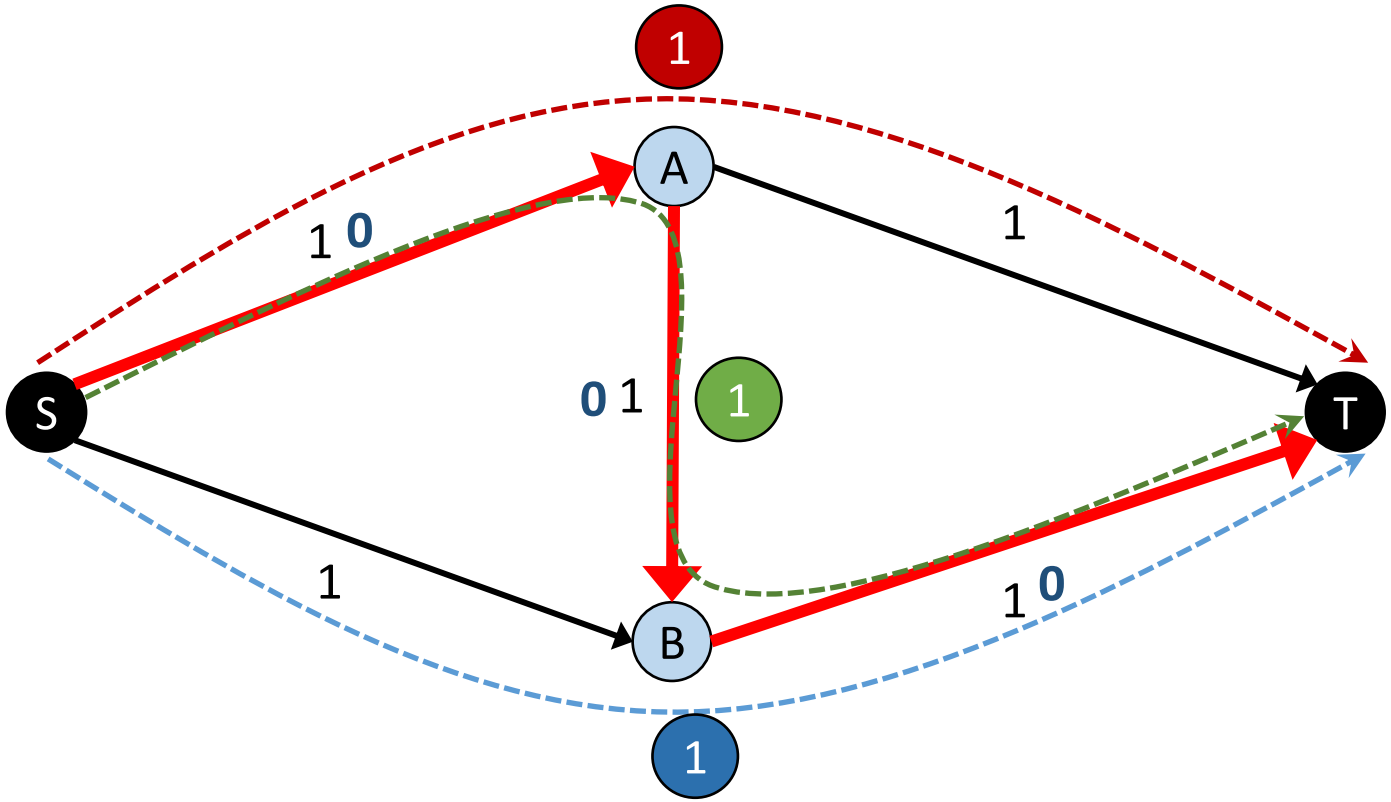


Max Flow – Problem with the Algorithm



The max flow clearly is of **size 2**

Max Flow – Problem with the Algorithm



If the greedy algorithm adds a flow of size 1 via the ***s* - *t*** path **s, a, b, t**

No ***s* - *t*** path in the remaining graph

Max Flow – Problem with the Algorithm

How can we resolve this issue?

Next Class

Max Flow – Problem with the Algorithm

How can we resolve this issue?

Next Class

Shepard: Dynamic Placement of Microservices in the Edge-Cloud Continuum

Farhan Asghar[1], Tehreem Fatima[1], Junaid Haroon Siddiqui[1], Naveed Anwar Bhatti[1], and Muhammad Hamad Alizai[1]

Lahore University of Management and Sciences LUMS, Lahore, Pakistan
{18030017,18030009, junaid.siddiqui, naveed.bhatti, hamad.alizai}@lums.edu.pk

Abstract. We present Shepard, an innovative microservice placement approach tailored for edge-assisted cloud infrastructures. Shepard dynamically migrates application services between the edge and cloud to harness optimal performance gains. This approach is structured around three core components: (1) a *resource manager* for monitoring available edge resources and the evolving demands of applications, (2) an *optimization module* that transposes the service placement dilemma into a labeled-graph cut challenge, aiming to identify the most advantageous cut given a set of parameters, and (3) a *deployment module* tasked with adjusting service placement in response to shifts in the optimal graph cut's position.
Our implementation of Shepard underwent rigorous testing in two distinct case studies. In the inaugural study, Shepard managed energy for a solar-driven edge within an agricultural IoT framework, resulting in a striking 79% elevation in service reliability and availability compared to a conventional static service placement strategy. For our subsequent study focusing on cost-effectiveness within a ride-hailing application, Shepard facilitated a substantial 45% slash in application deployment expenses, all the while maintaining comparable performance levels to a standard dynamic service placement technique.

Keywords: Edge Computing · Microservices-based Architecture · Dynamic Resource Management.

1 Introduction

The evolution of edge computing has created the need for new ways of structuring and deploying application services. An edge is not just a homogeneous extension of cloud resources that can scale up application services on demand, but a precious compute resource near the data source. Intelligently managing this limited yet expensive compute resource is imperative to enhance application performance or extend the advantages of edge to multiple applications (or tenants). As such, deciding what part of the application logic goes into the edge and what stays in the cloud has become a key DevOps challenge.

Thanks a lot

If you are taking a Nap, **wake up**.....***Lecture OVER***