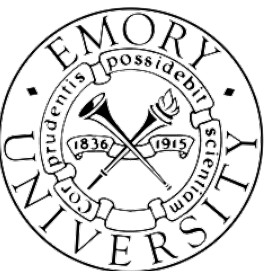
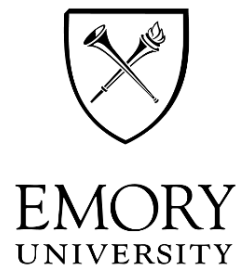


Network Flow

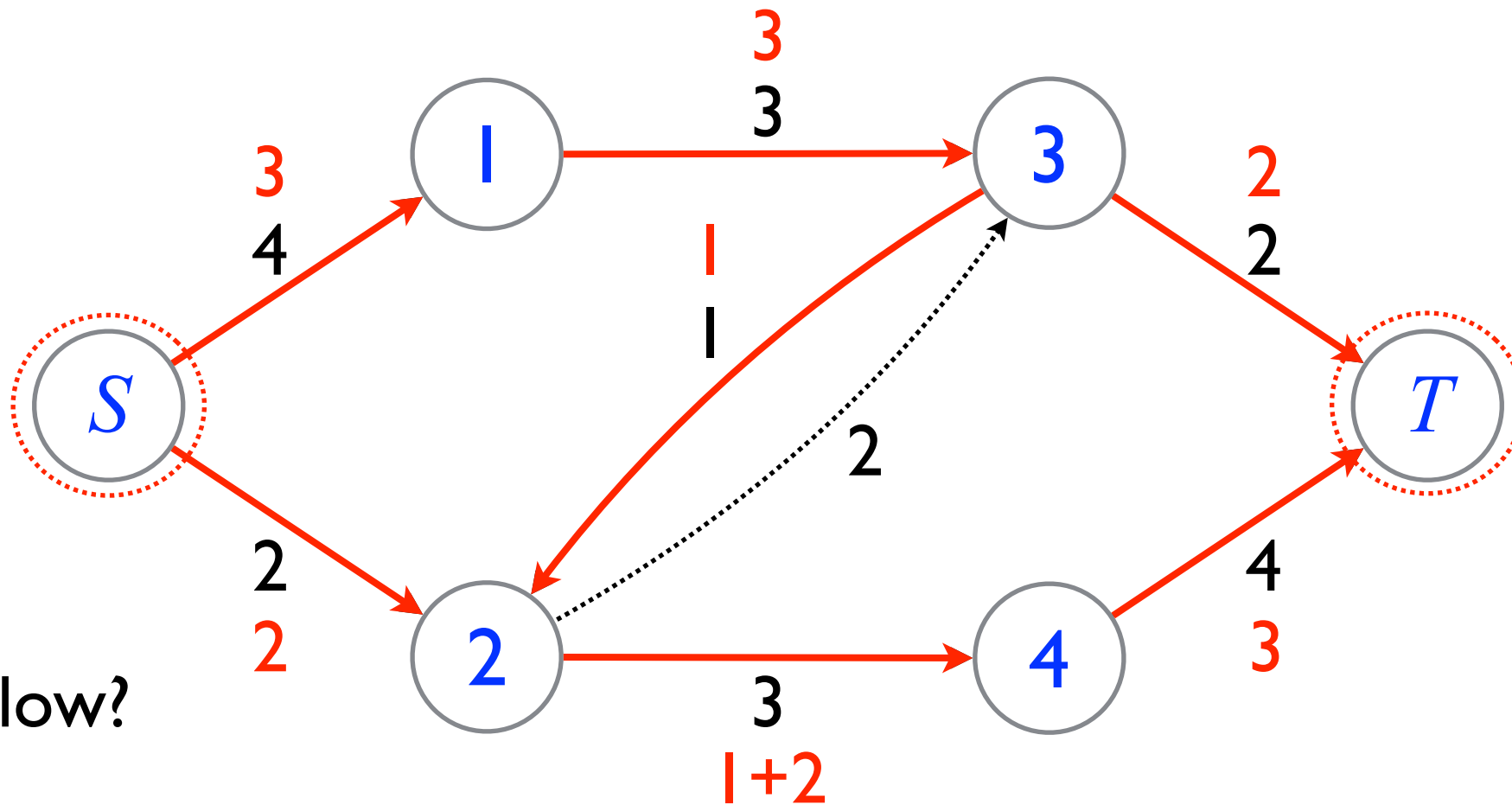
Data Structures and Algorithms

Emory University

Jinho D. Choi



Network Flow



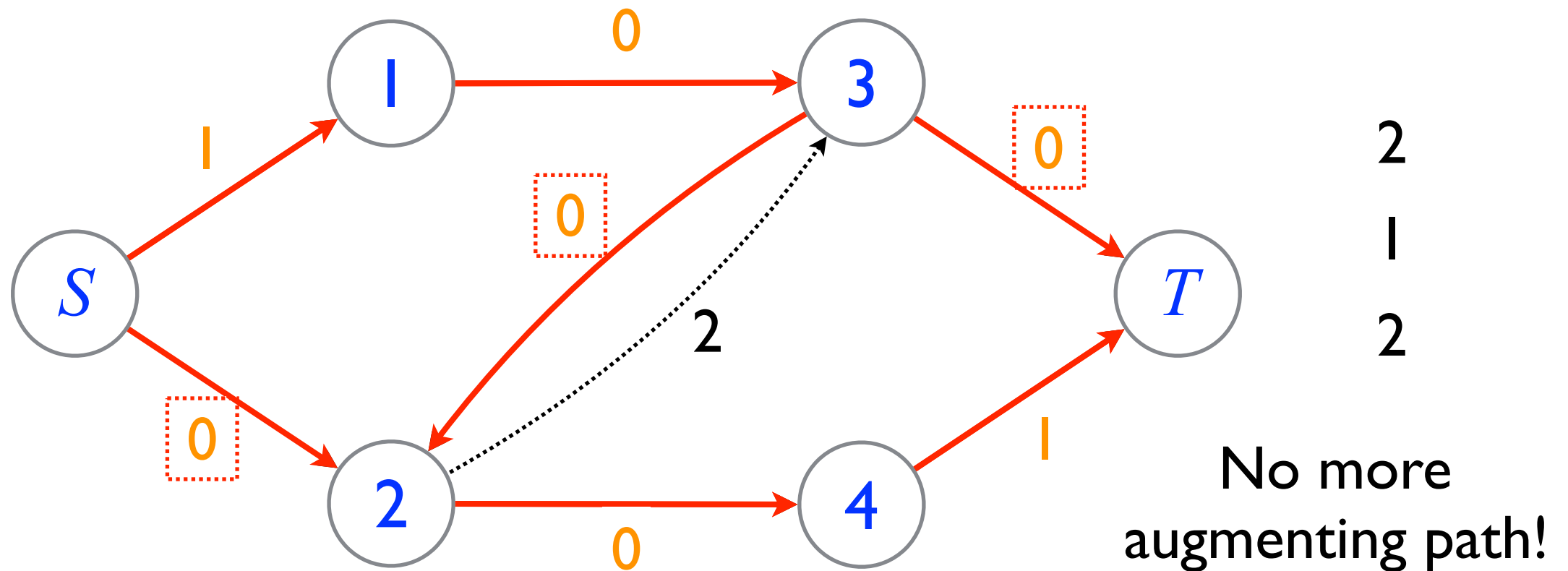
Each edge e is associated with a capacity c .

Push as much flow f as possible from S to T .

$$f(e) \leq c(e)$$

$$\sum_u f(u, v) = \sum_w f(v, w), \text{ where } v \notin \{S, T\}$$

Ford-Fulkerson Algorithm



Find a path p from S to T , where $\forall e \in p. r(e) = c(e) - f(e) > 0$.

$$\forall e \in p. c(e) = c(e) - \min(f(p)).$$

$$\text{MaxFlow} = \text{MaxFlow} + \min(f(p)).$$

MaxFlow Class

```
public class MaxFlow
{
    private Map<Edge, Double> m_flows;
    private double d_maxFlow;

    public MaxFlow(Graph graph)
    {
        init(graph);
    }

    public void init(Graph graph)
    {
        m_flows = new HashMap<>();
        d_maxFlow = 0;

        for (Edge edge : graph.getAllEdges())
            m_flows.put(edge, 0d);
    }
}
```

MaxFlow Class

```
public void updateResidual(List<Edge> path, double flow)
{
    for (Edge edge : path) updateResidual(edge, flow);
    d_maxFlow += flow;
}
```

```
public void updateResidual(Edge edge, double flow)
{
    Double prev = m_flows.get(edge);
    if (prev == null) prev = 0d;
    m_flows.put(edge, prev + flow);
}
```

```
public double getResidual(Edge edge)
{
    return edge.getWeight() - m_flows.get(edge);
}
```

```
public double getMaxFlow()
{
    return d_maxFlow;
}
```

FordFulkerson Class

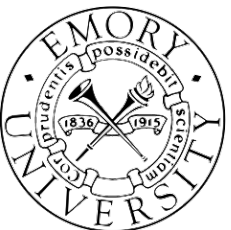
```
private Subgraph getAugmentingPath(Graph graph, MaxFlow mf,
                                   Subgraph sub, int source, int target)
{
    if (source == target) return sub;
    Subgraph tmp;

    for (Edge edge : graph.getIncomingEdges(target))
    {
        if (sub.contains(edge.getSource())) continue; // cycle
        if (mf.getResidual(edge) <= 0) continue; // no residual
        tmp = new Subgraph(sub);
        tmp.addEdge(edge);

        tmp = getAugmentingPath(graph, mf, tmp, source, edge.getSource());
        if (tmp != null) return tmp;
    }

    return null;
}
```

Complexity?



FordFulkerson Class

```
public MaxFlow getMaximumFlow(Graph graph, int source, int target)
{
    MaxFlow mf = new MaxFlow(graph);
    Subgraph sub = new Subgraph();
    double min;

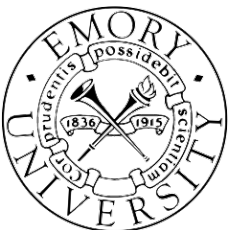
    while ((sub = getAugmentingPath(graph, mf, sub, source, target)) != null)
    {
        min = getMin(mf, sub.getEdges());
        mf.updateResidual(sub.getEdges(), min);
    }

    return mf;
}

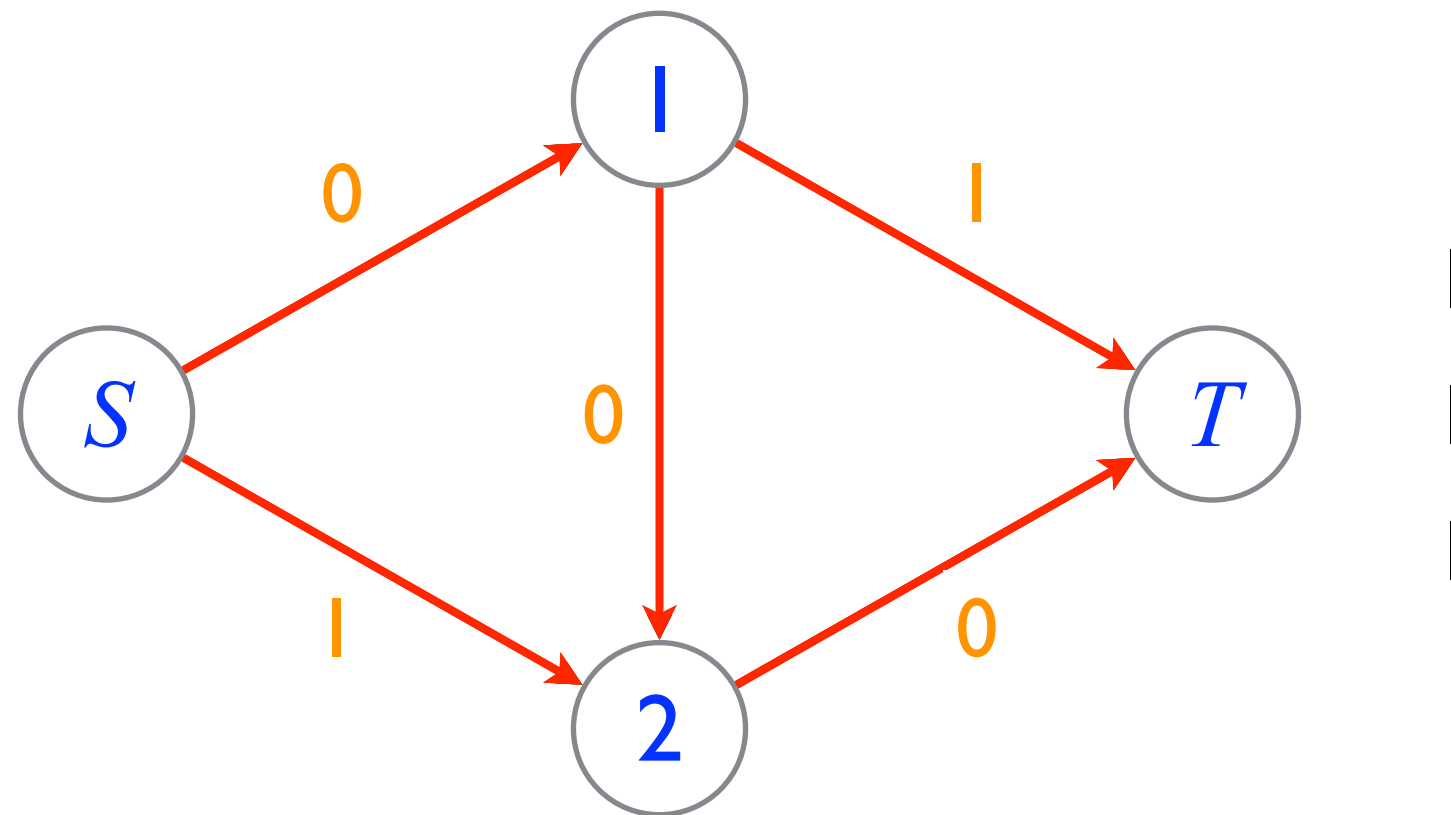
private double getMin(MaxFlow mf, List<Edge> path)
{
    double min = mf.getResidual(path.get(0));

    for (int i=1; i<path.size(); i++)
        min = Math.min(min, mf.getResidual(path.get(i)));

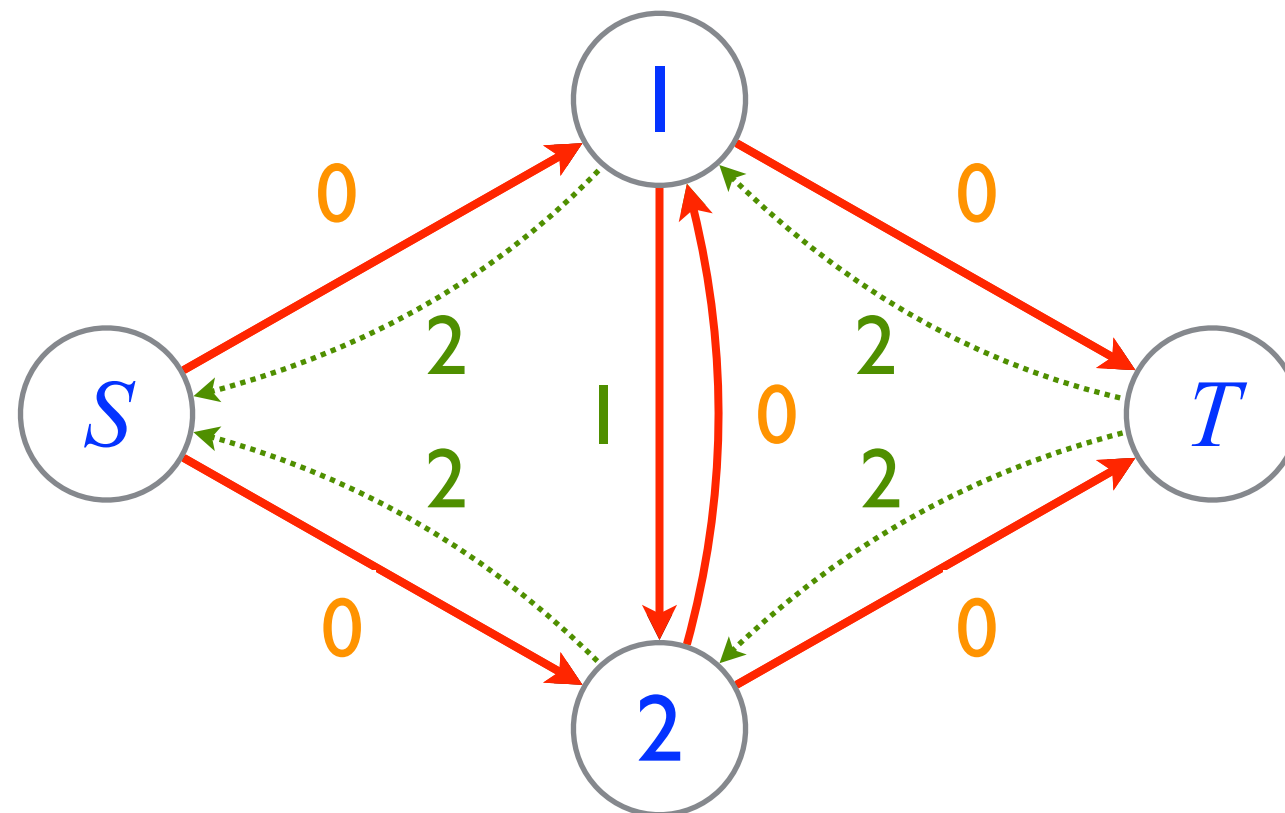
    return min;
}
```



Ford-Fulkerson: Backward Pushing



Ford-Fulkerson: Backward Pushing



```

protected void updateBackward(Graph graph, Subgraph sub, MaxFlow mf, double min)
{
    boolean found;

    for (Edge edge : sub.getEdges())
    {
        found = false;

        for (Edge rEdge : graph.getIncomingEdges(edge.getSource()))
        {
            if (rEdge.getSource() == edge.getTarget())
            {
                mf.updateResidual(rEdge, -min);
                found = true; break;
            }
        }

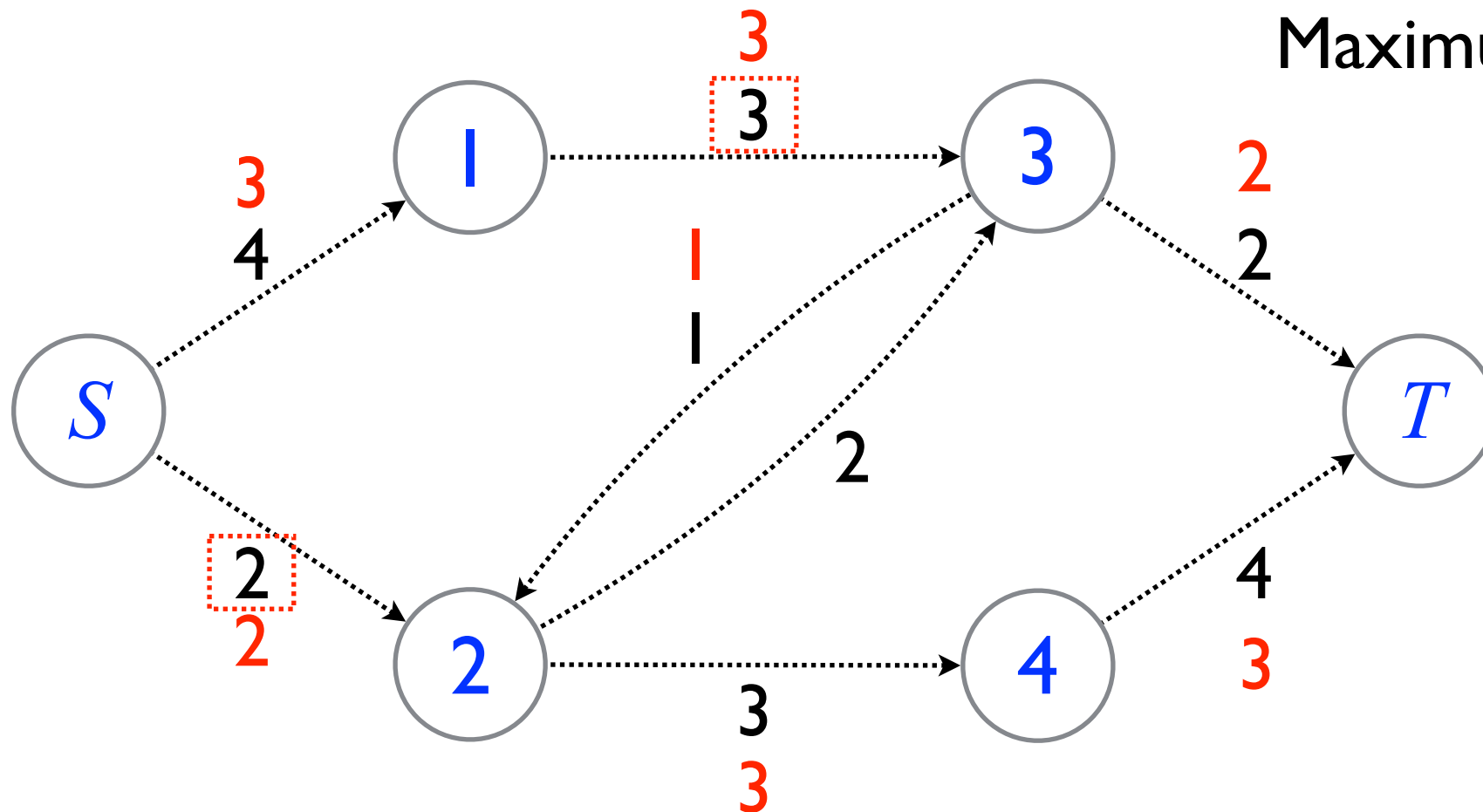
        if (!found)
        {
            Edge rEdge = graph.setDirectedEdge
                (edge.getTarget(), edge.getSource(), edge.getWeight());
            mf.updateResidual(rEdge, -min);
        }
    }
}

```

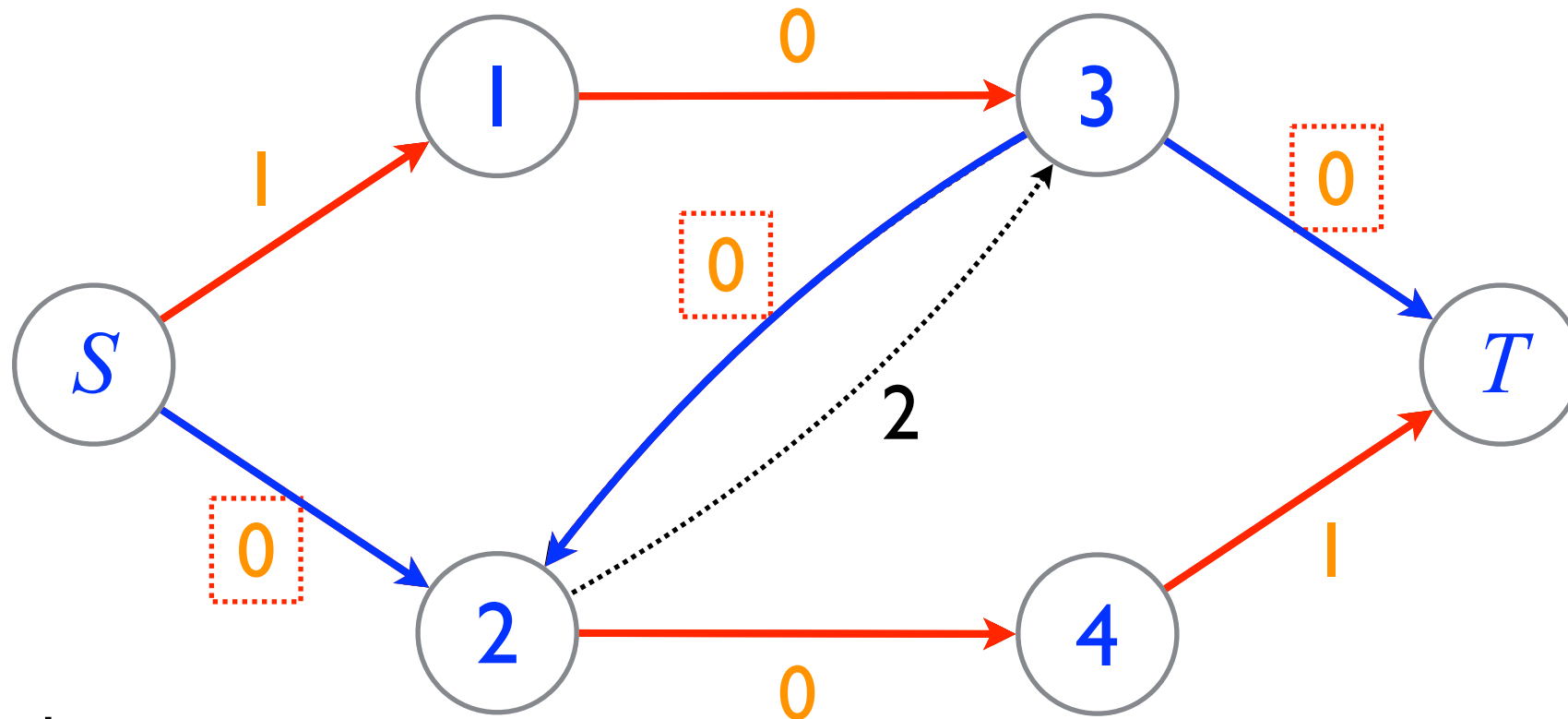
Max-Flow Min-Cut Theorem

- S - T cut
 - A set of edges whose removal disconnects S to T .
 - The capacity of a cut = $\sum c(e)$, $\forall e$ in the cut.

Minimum cut
vs.
Maximum flow



Finding Min-Cut



- Algorithm
 1. Find a path p from S to T , remove any edge e in p , and put e to a set C .
 2. Repeat #1 until no path is found, and return C .
- How can we find a min-cut?
 - ─ Instead of removing any edge in #1, remove an edge with the **minimum** original (not residual) weight.

Min-Cut vs. Max-Flow

- Lemma 1
 - There is no **extra edge** in the min-cut.
 - Prove?
- Lemma 2
 - All edges in the min-cut must be parts of **augmenting paths**.
 - Prove?
- Lemma 3
 - Each edge in the min-cut is the **minimum weighted edge** in each augmenting path.
 - Prove?

Network Flow Using Simplex

Max-Flow

Maximize:

$$x_2 + x_4$$

Subject to:

$$x_1 \leq 4$$

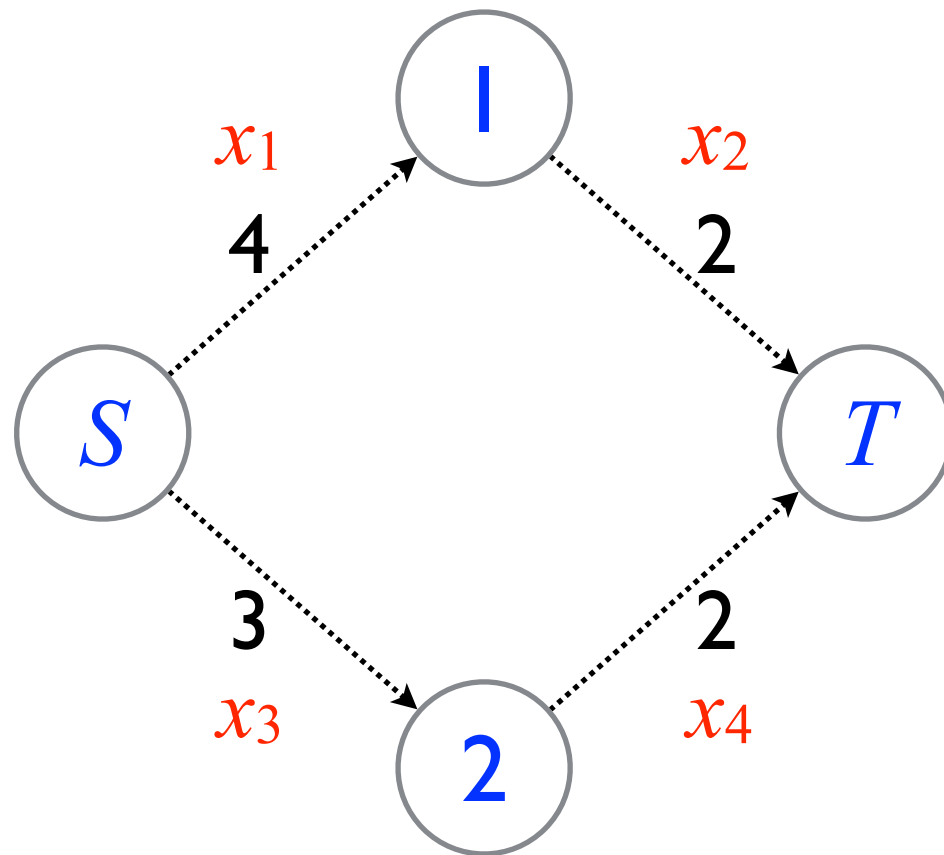
$$x_2 \leq 2$$

$$x_3 \leq 3$$

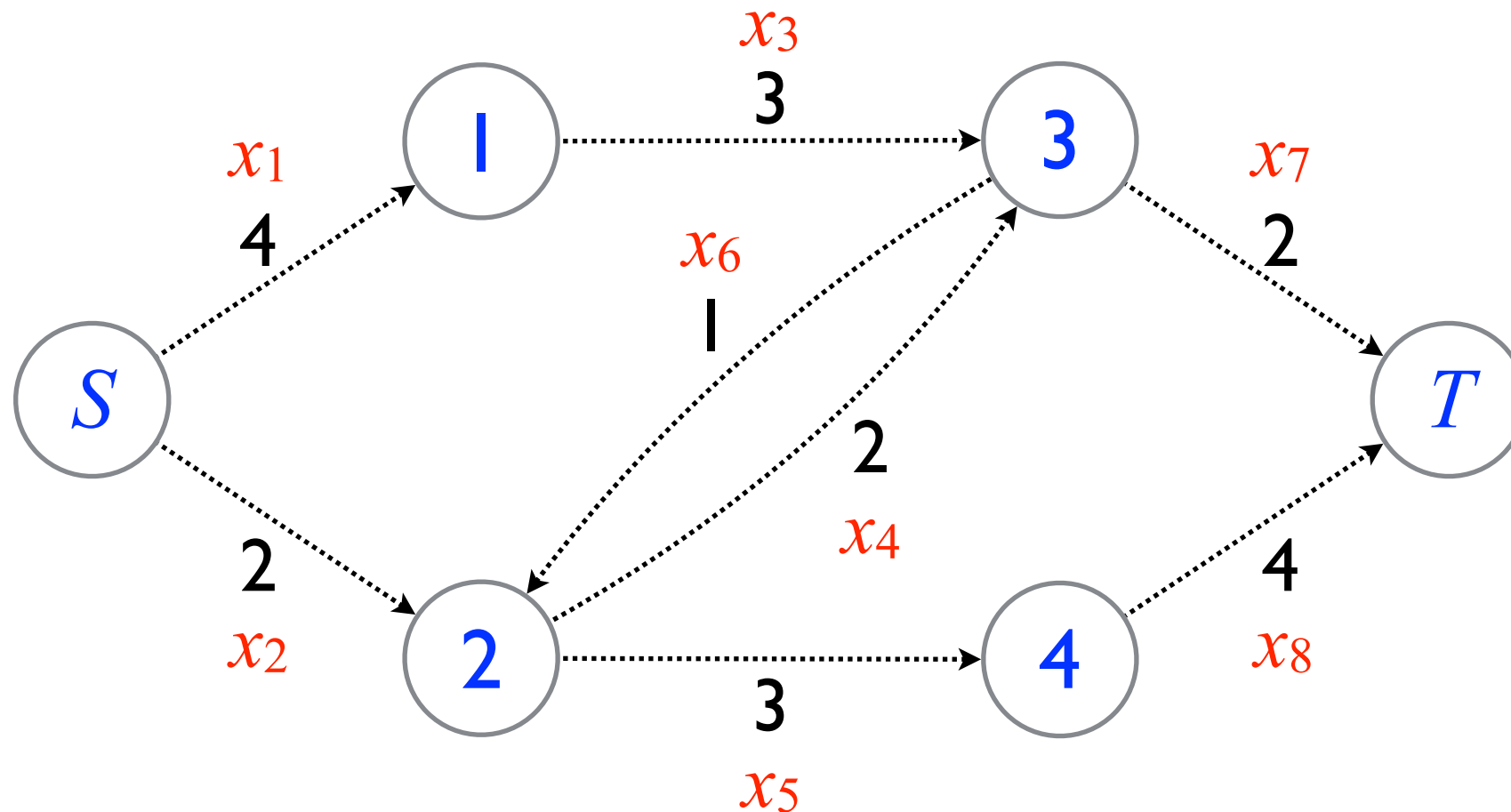
$$x_4 \leq 2$$

$$x_2 - x_1 \leq 0$$

$$x_4 - x_3 \leq 0$$



Network Flow Using Simplex



Maximize:

$$x_7 + x_8$$

Subject to:

$$x_1 \leq 4$$

$$x_2 \leq 2$$

$$x_3 \leq 3$$

$$x_4 \leq 2$$

$$x_5 \leq 3$$

$$x_6 \leq 1$$

$$x_7 \leq 2$$

$$x_8 \leq 4$$

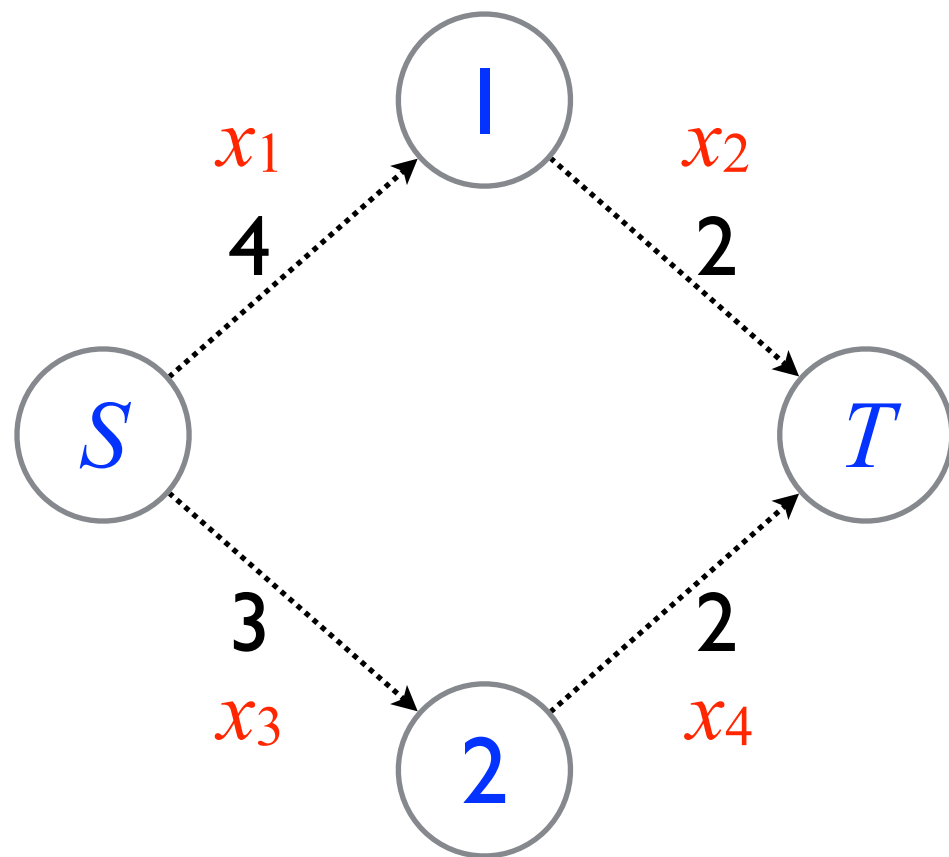
$$x_3 - x_1 \leq 0$$

$$x_4 + x_5 - x_2 - x_6 \leq 0$$

$$x_6 + x_7 - x_3 - x_4 \leq 0$$

$$x_8 - x_5 \leq 0$$

Network Flow Using Simplex



Max-Flow

Maximize:

$$x_2 + x_4$$

Subject to:

$$x_1 \leq 4$$

$$x_2 \leq 2$$

$$x_3 \leq 3$$

$$x_4 \leq 2$$

$$x_2 - x_1 \leq 0$$

$$x_4 - x_3 \leq 0$$

Min-Cut

Minimize:

$$4x_1 + 2x_2 + 3x_3 + 2x_4$$

Subject to:

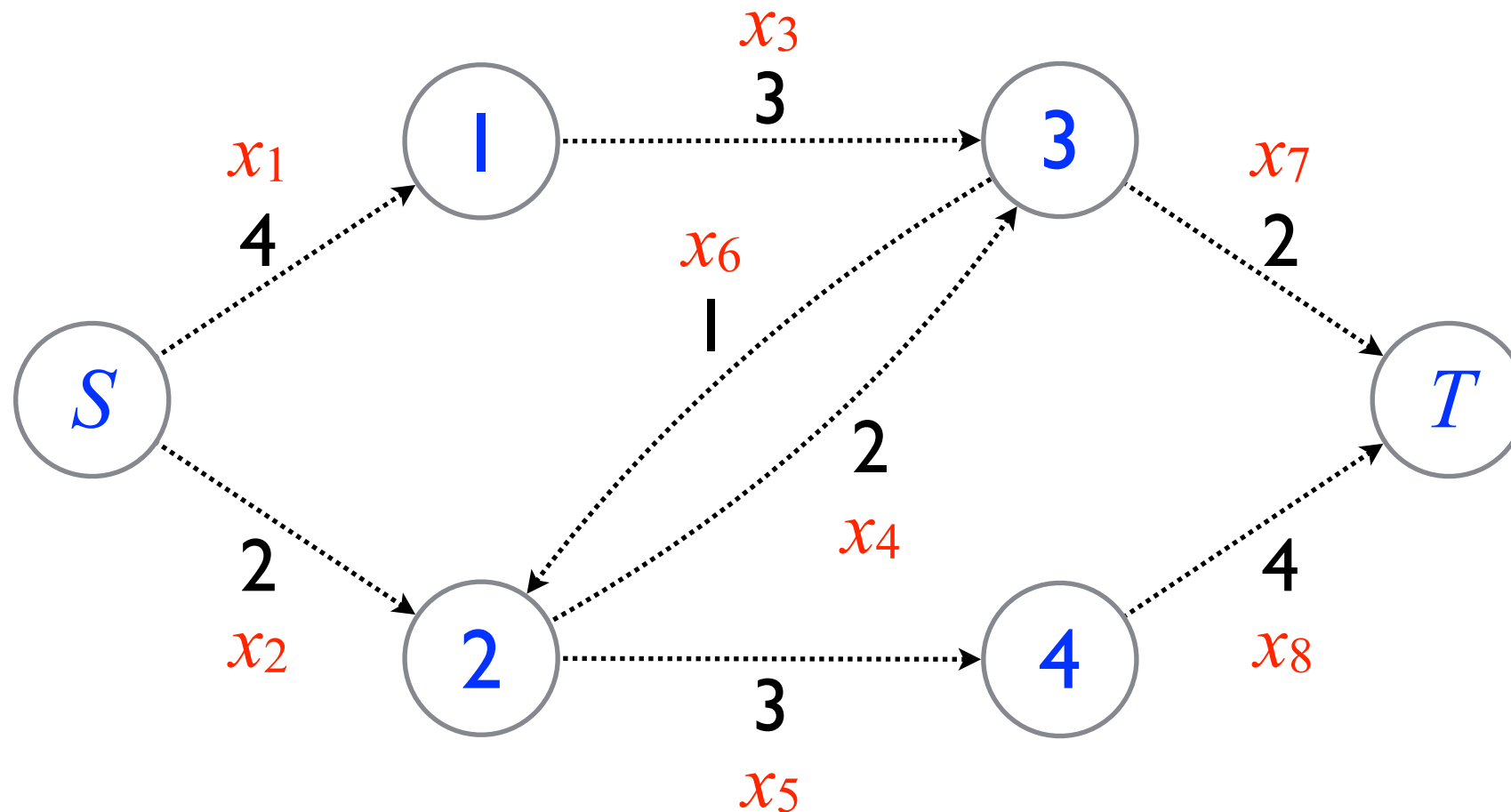
$$x_1 + y_1 \geq 1$$

$$x_3 + y_3 \geq 1$$

$$x_2 - y_1 \geq 0$$

$$x_4 - y_3 \geq 0$$

Network Flow Using Simplex



Minimize:

$$4x_1 + 2x_2 + 3x_3 + 2x_4 + 3x_5 + x_6 + 2x_7 + 4x_8$$

Subject to:

$$\begin{array}{ll} x_1 + y_1 \geq 1 & x_5 - y_2 + y_4 \geq 0 \\ x_2 + y_2 \geq 1 & x_6 - y_3 + y_2 \geq 0 \\ x_3 - y_1 + y_3 \geq 0 & x_7 - y_3 \geq 0 \\ x_4 - y_2 + y_3 \geq 0 & x_8 - y_4 \geq 0 \end{array}$$

