Network Algorithm UE709

Rabih AMHAZ

Head of Digital department at Icam Strasbourg-Europe Campus
Researcher in AI and Blockchain at ICube Laboratory, CSTB team

February 13, 2024

Maximum Flow : Question

With an infinit input source, how much "flow" can we push through the network given that each edge has a certain capacity?

Edges capacity : can receive a certain amount of flow.

Maximum Flow

Maximum Flow : Question

With an infinit input source, how much "flow" can we push through the network given that each edge has a certain capacity?

Edges capacity : can receive a certain amount of flow.

The key takeaway here is that the Ford-Fulkerson method does not specify how to actually find the augmenting paths.
This is where **optimizations** come into the game.
ex: using a DFS to find augmented paths takes $O(Ef)$

Edmonds-Karp algorithm

The Edmonds-Karp algorithm uses a Bredth First Search (BFS) to find the augmenting paths which yields an arguably better time complexity of $O(VE^2)$.
The major difference in this approach is that the time complexity no longer depends on the capacity value of any edge.
Strongly polynomial

- as a method of augmentation which repeatedly finds the shortest augmenting path from s to t in terms of the **number of edges** used in each iteration.

The key takeaway here is that the Ford-Fulkerson method does not specify how to actually find the augmenting paths.
This is where **optimizations** come into the game.
ex: using a DFS to find augmented paths takes $O(Ef)$

Edmonds-Karp algorithm

The Edmonds-Karp algorithm uses a Bredth First Search (BFS) to find the augmenting paths which yields an arguably better time complexity of $O(VE^2)$.
The major difference in this approach is that the time complexity no longer depends on the capacity value of any edge.
Strongly polynomial

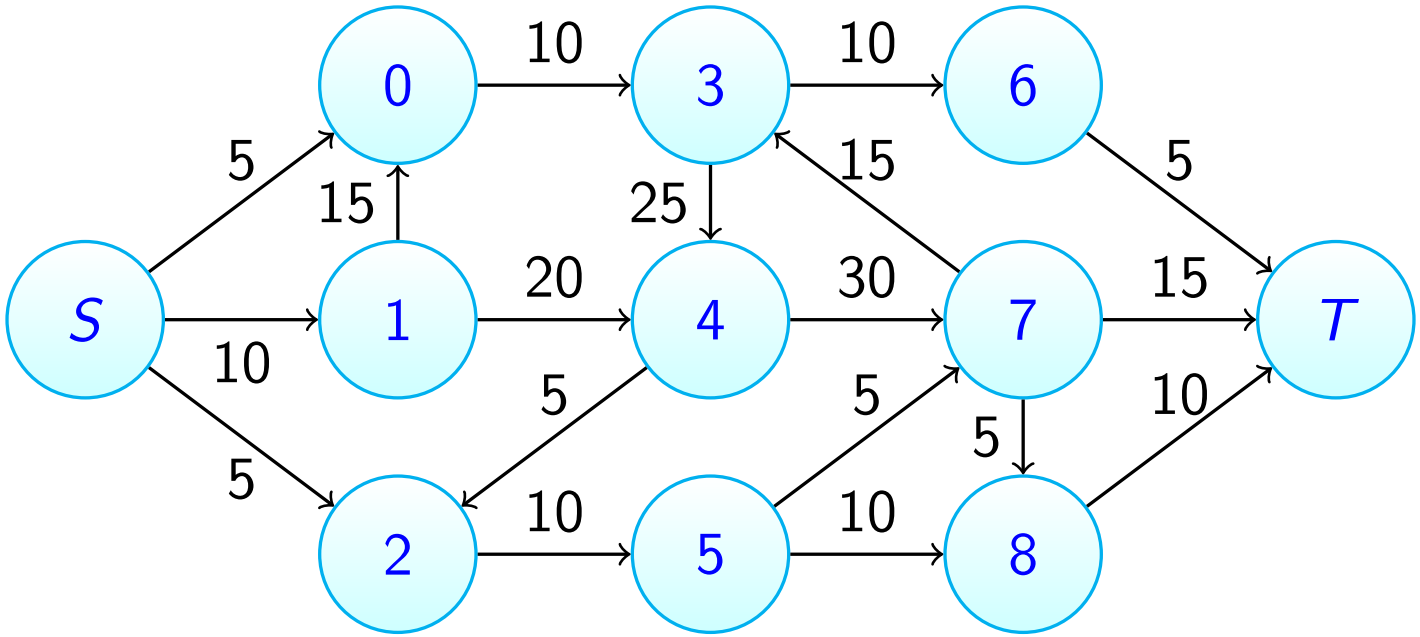
- as a method of augmentation which repeatedly finds the shortest augmenting path from s to t in terms of the **number of edges** used in each iteration.

The key takeaway here is that the Ford-Fulkerson method does not specify how to actually find the augmenting paths.
This is where **optimizations** come into the game.
ex: using a DFS to find augmented paths takes $O(Ef)$

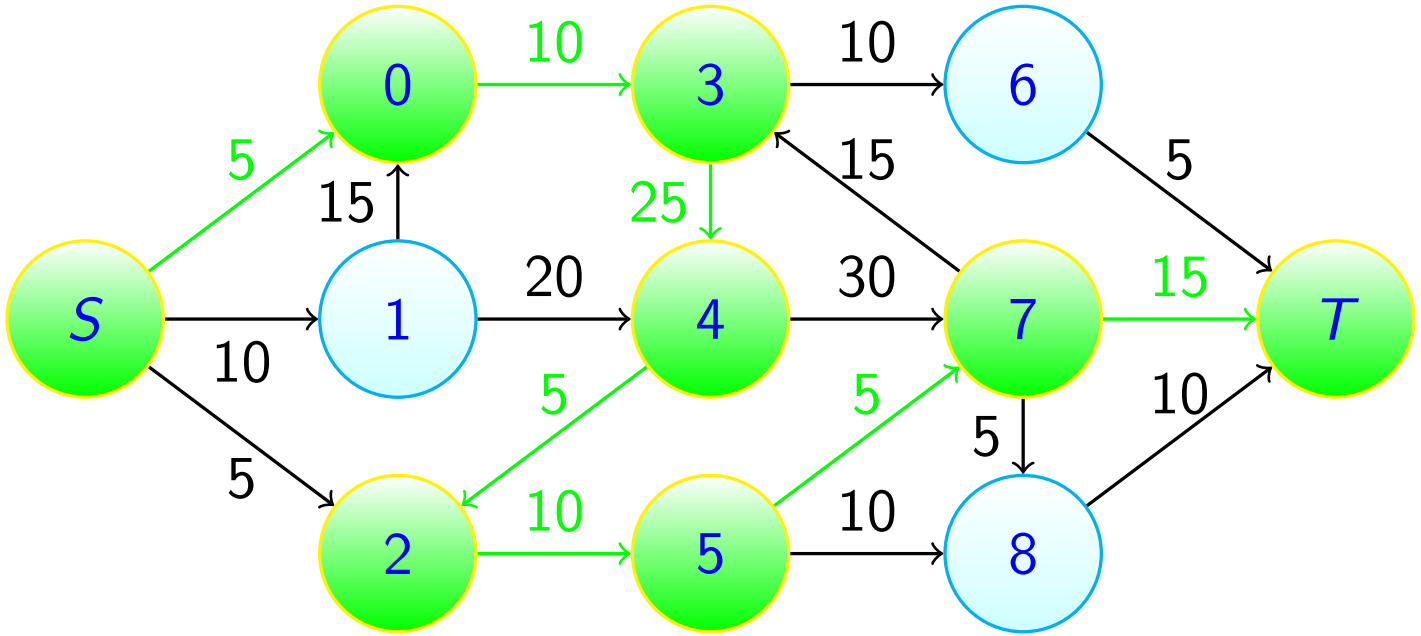
Edmonds-Karp algorithm

The Edmonds-Karp algorithm uses a Bredth First Search (BFS) to find the augmenting paths which yields an arguably better time complexity of $O(VE^2)$.
The major difference in this approach is that the time complexity no longer depends on the capacity value of any edge.
Strongly polynomial

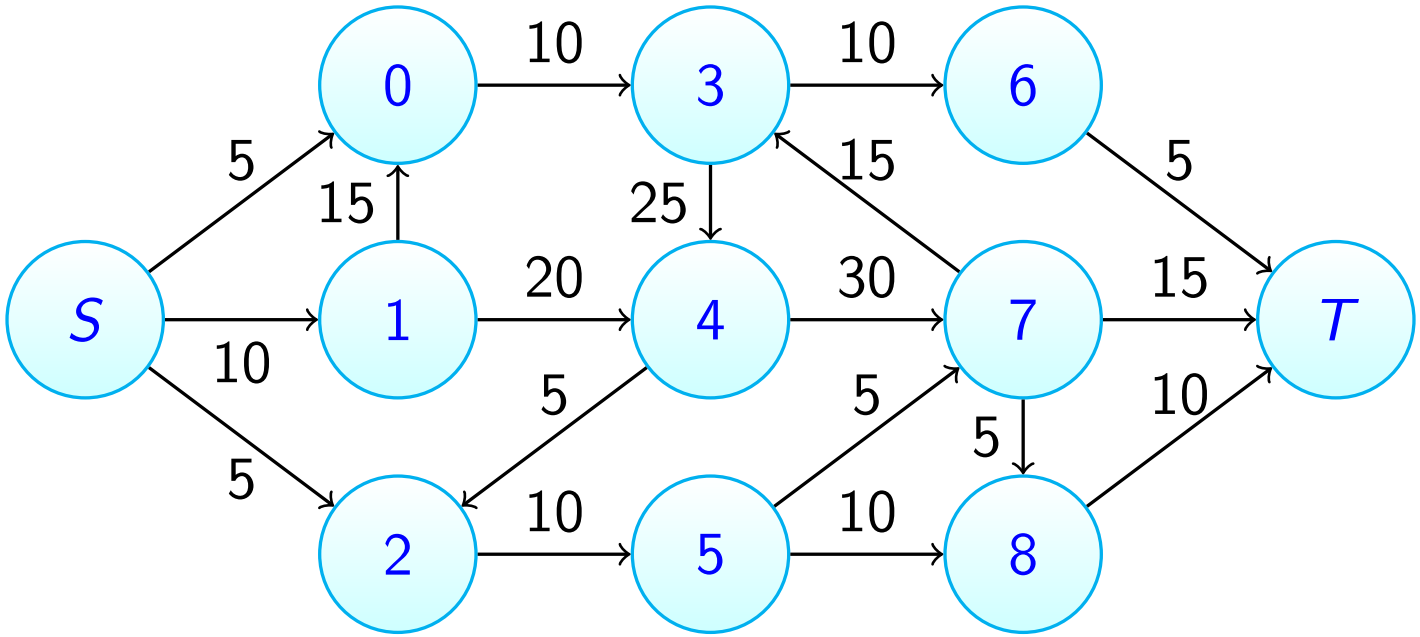
- as a method of augmentation which repeatedly finds the shortest augmenting path from s to t in terms of the **number of edges** used in each iteration.

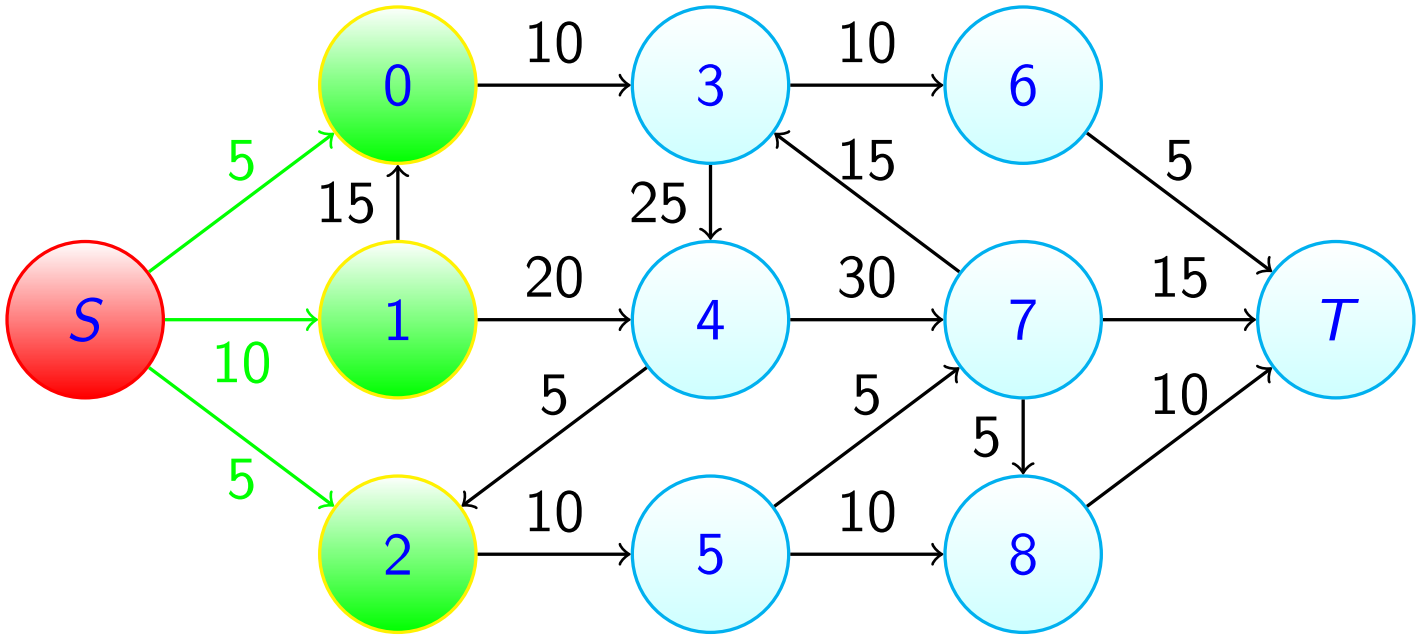

0
3
6
S
1
4
7
T
2
5
8
10
10
5
15
25
15
5
20
30
15
10
5
5
10
5
10
10

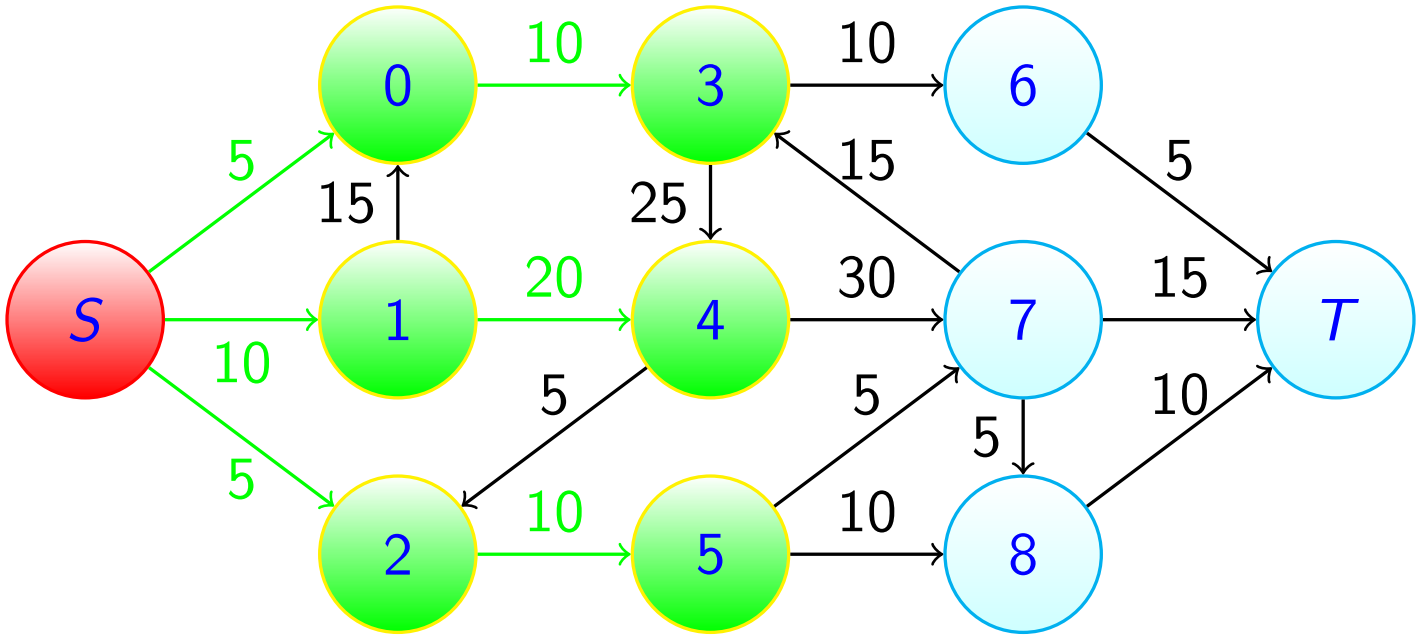
If we use Ford-Fulkerson and we use DFS, we can have a very long augmented path.



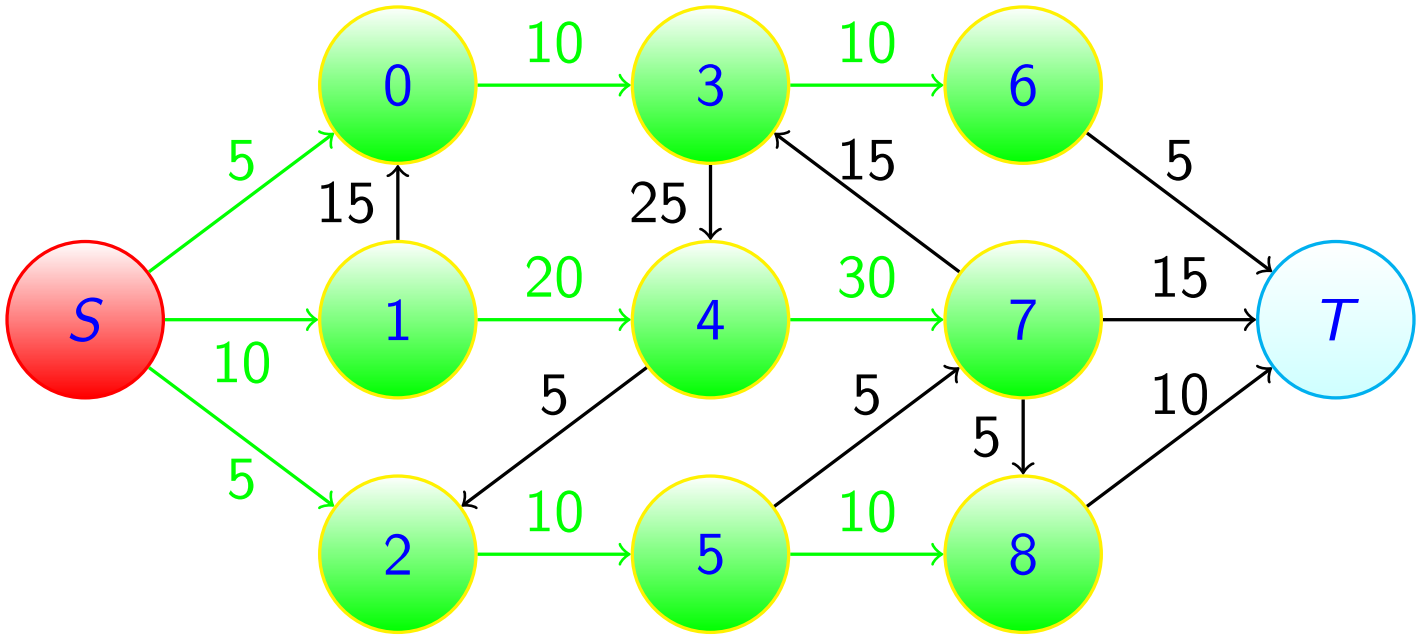
This is the big difference with Edmonds-Karps that use **BFS** so all paths will be smallest in number of edges.

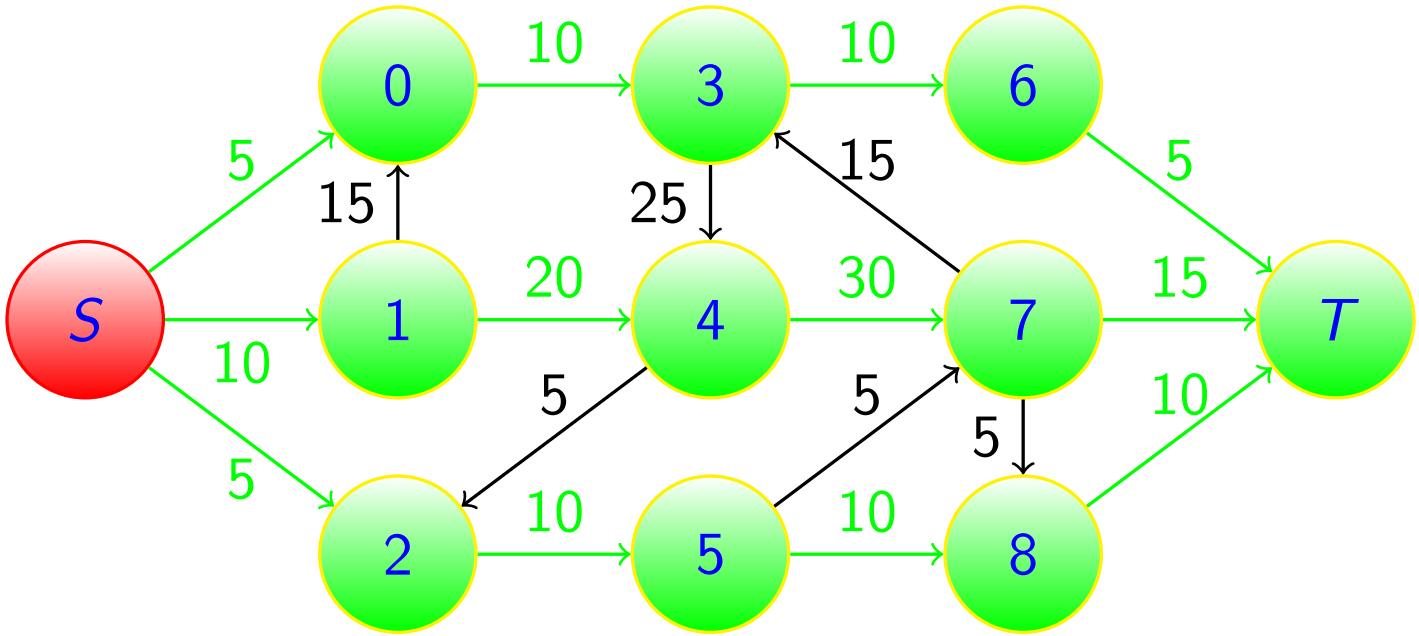

0
3
6
S
1
4
7
T
2
5
8
10
10
5
15
25
15
5
20
30
15
10
5
5
10
5
10
10

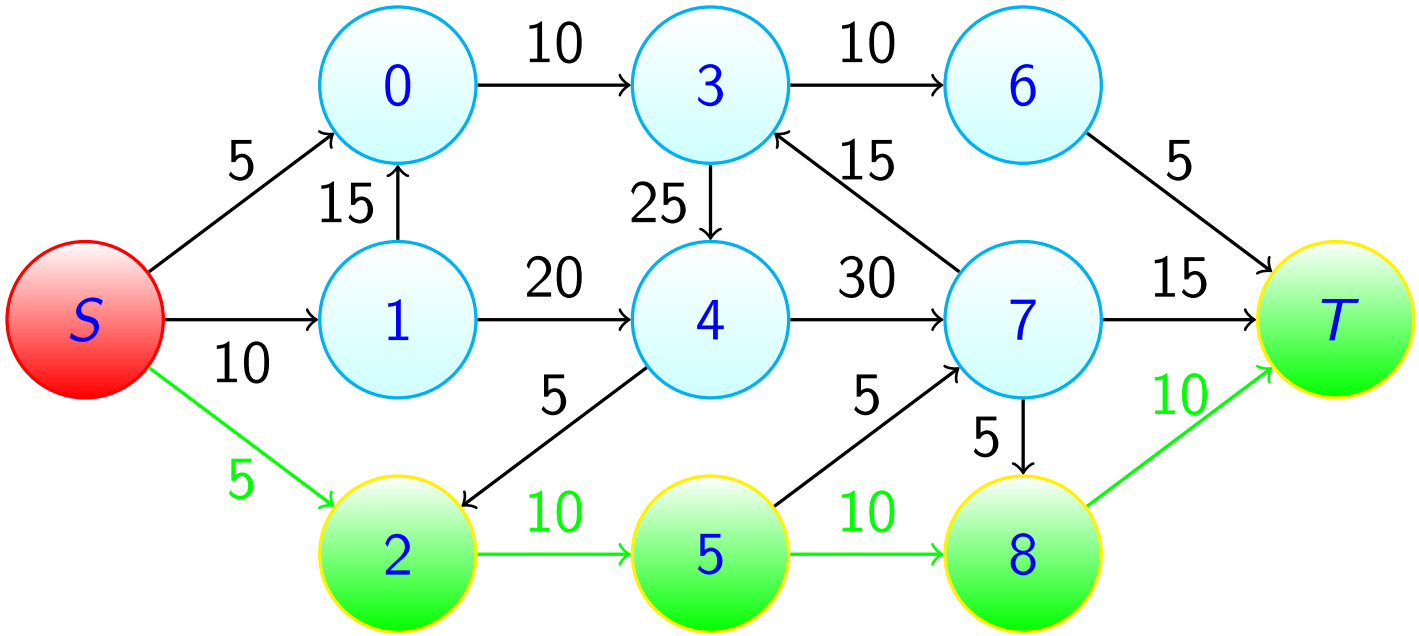

S
0
1
2
3
4
5
6
7
8
T
5
10
5
10
15
25
10
20
5
10
15
30
5
5
10
10
15

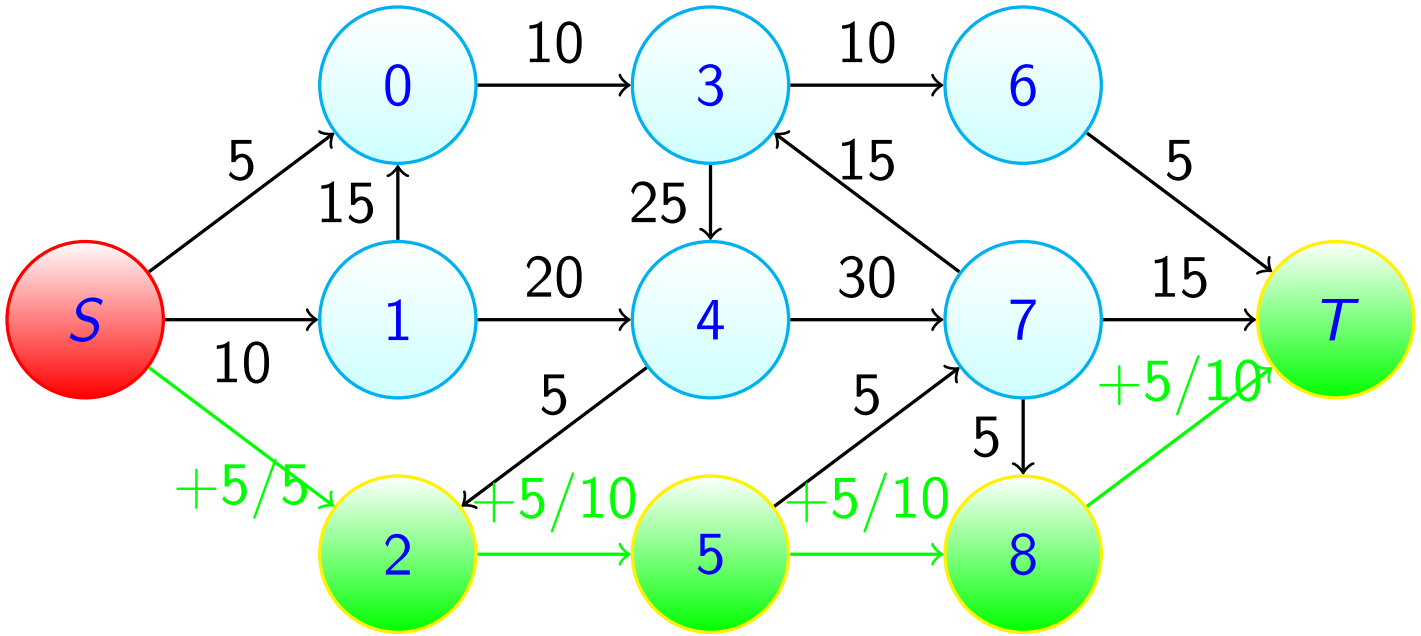

S
0
1
2
3
4
5
6
7
8
T
5
10
15
10
10
15
25
20
30
5
15
5
5
10
5
10

MaxFlow: Edmonds-Karp

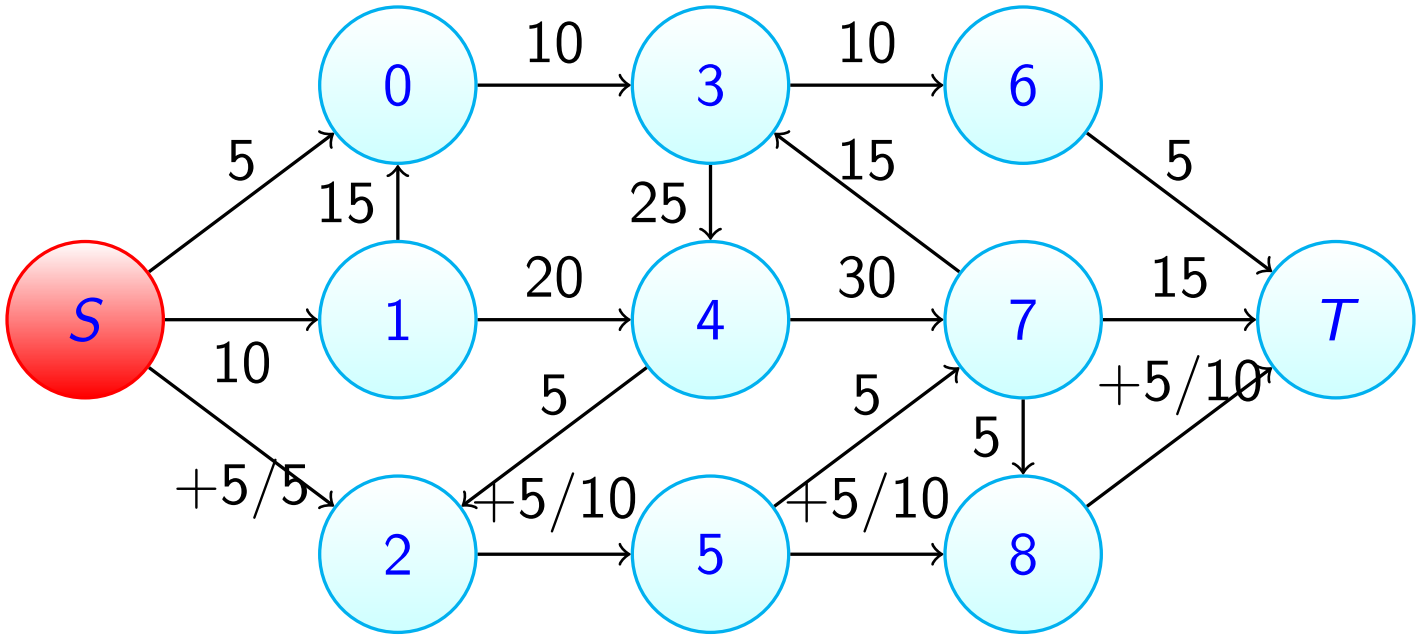



S
0
1
2
3
4
5
6
7
8
T
5
10
5
10
15
20
25
10
5
15
30
5
5
10
10
10
15

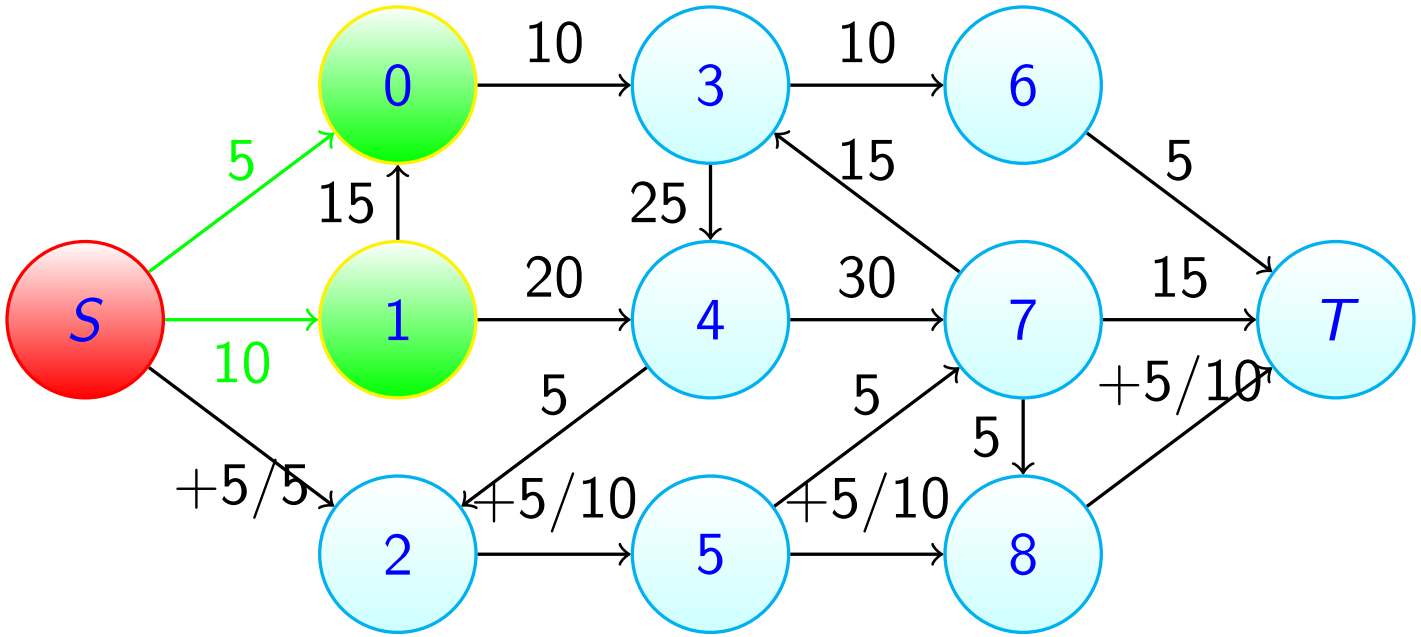

S
0
1
2
3
4
5
6
7
8
T
10
5
15
10
25
20
5
10
15
30
5
5
5
10
10
10


0
3
6
S
1
4
7
T
2
5
8
10
10
5
15
25
15
5
20
30
15
10
5
5
5
+5/10
+5/5
+5/10
+5/10

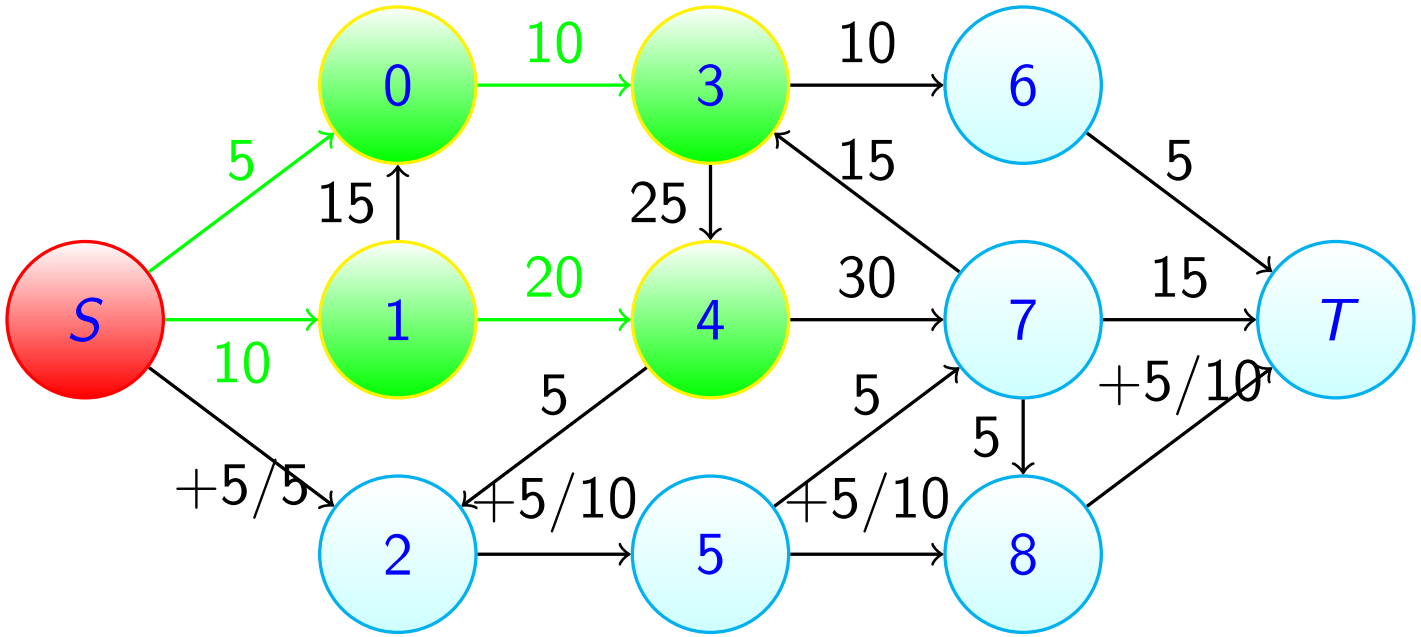
MaxFlow: Edmonds-Karp

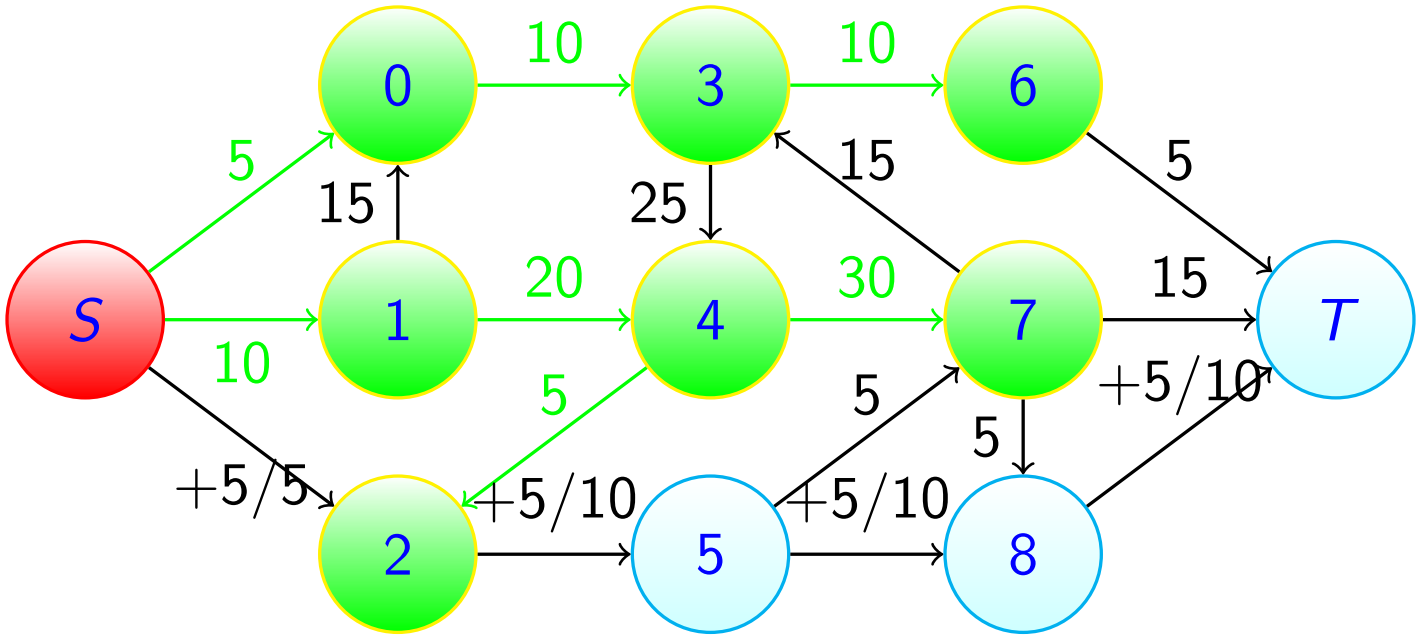


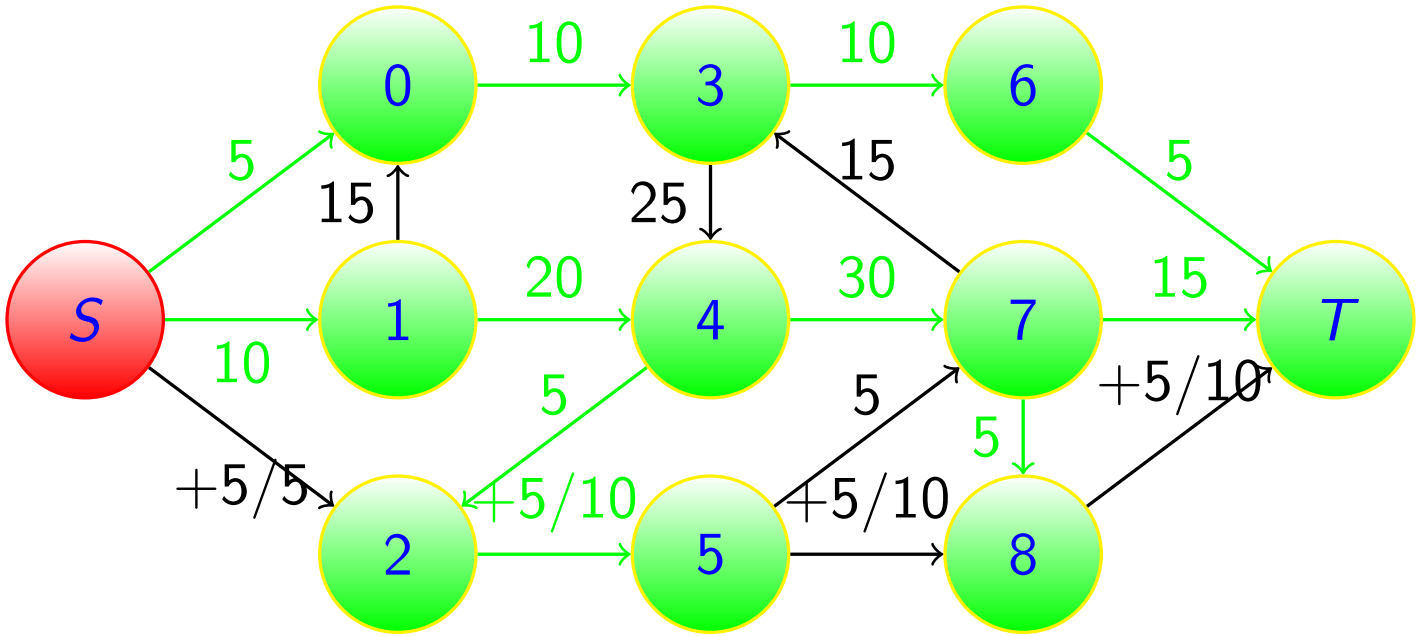
MaxFlow: Edmonds-Karp



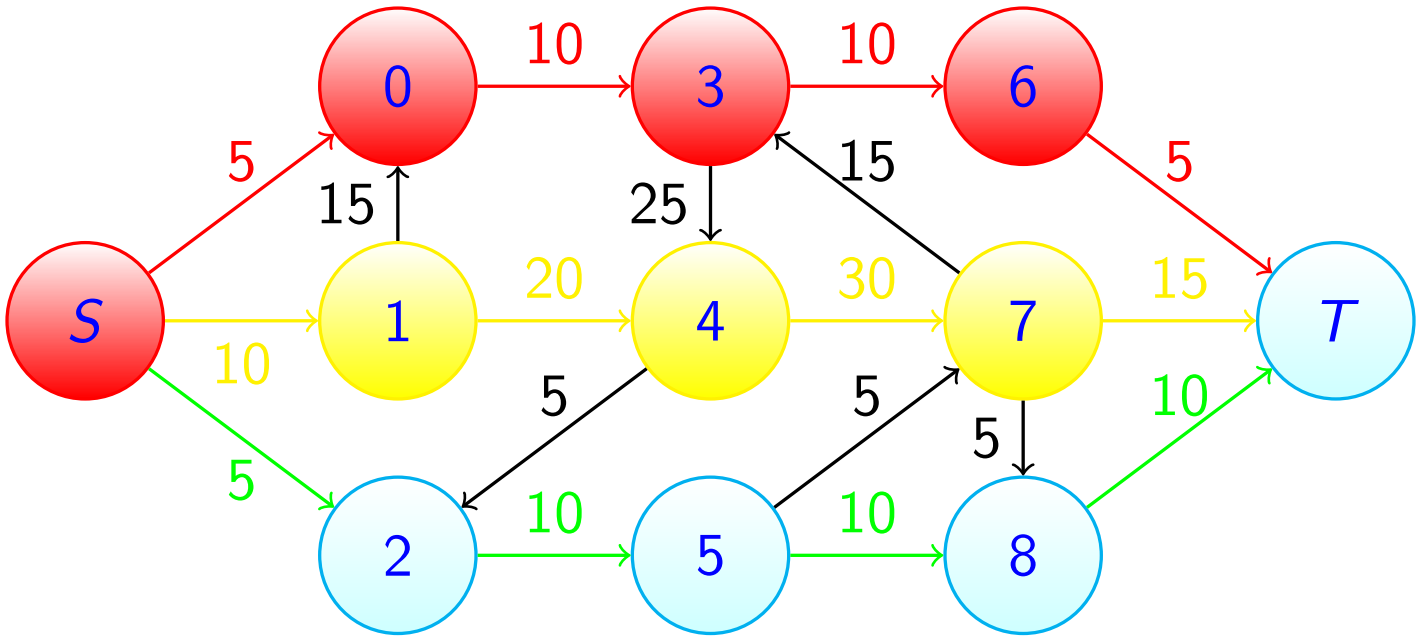
MaxFlow: Edmonds-Karp




0
3
6
S
1
4
7
T
2
5
8
10
10
5
15
25
15
5
20
30
15
10
5
5
+5/5
+5/10
+5/10
+5/10


S
0
1
2
3
4
5
6
7
8
T
10
5
15
10
20
25
30
15
5
10
5
5
+5/5
+5/10
+5/10
+5/10

MaxFlow: Edmonds-Karp



$$\text{MaxFlow} = \sum bottleneckValues = 5 + 10 + 5 = 20$$

MaxFlow: Dinic's

Strongly polynomial max flow algorithm. $O(V^2E)$
Extremly fast **and works better** on biprtite graph. $O(\sqrt{VE})$
Yefin Dinitz 1969

Analogy: I call you I have 1% battery

coffee shop a few streets east.

You dont know the coffee shop. How can you get there?

Se we use level graph.

MaxFlow: Dinic's

Strongly polynomial max flow algorithm. $O(V^2E)$
Extremly fast **and works better** on biprtite graph. $O(\sqrt{VE})$
Yefin Dinitz 1969

Analogy: I call you I have 1% battery

coffee shop a few streets east.

You dont know the coffee shop. How can you get there?
Se we use level graph.

MaxFlow: Dinic's

Strongly polynomial max flow algorithm. $O(V^2E)$
Extremly fast **and works better** on biprtite graph. $O(\sqrt{VE})$
Yefin Dinitz 1969

Analogy: I call you I have 1% battery

coffee shop a few streets east.

You dont know the coffee shop. How can you get there?

Se we use level graph.

MaxFlow: Dinic's

Strongly polynomial max flow algorithm. $O(V^2E)$
Extremly fast **and works better** on biprtite graph. $O(\sqrt{VE})$
Yefin Dinitz 1969

Analogy: I call you I have 1% battery

coffee shop a few streets east.

You dont know the coffee shop. How can you get there?
Se we use level graph.

MaxFlow: Dinic's

Strongly polynomial max flow algorithm. $O(V^2E)$
Extremly fast **and works better** on biprtite graph. $O(\sqrt{VE})$

1. Construct a level graph by doing a BFS from source to label all the levels of the current flow graph.
2. If the sink was never reached while building the level graph, then stop and return max flow.
3. Using only valid edges in the level graph, do multiple DFSs from s->t until a blocking flow is reached, and sum over the bottleneck values of all the augmenting paths found to calculate the max flow.

MaxFlow: Dinic's

Strongly polynomial max flow algorithm. $O(V^2E)$
Extremly fast **and works better** on biprtite graph. $O(\sqrt{VE})$

1. Construct a level graph by doing a BFS from source to label all the levels of the current flow graph.
2. If the sink was never reached while building the level graph, then stop and return max flow.
3. Using only valid edges in the level graph, do multiple DFSs from s->t until a blocking flow is reached, and sum over the bottleneck values of all the augmenting paths found to calculate the max flow.

MaxFlow: Dinic's

Strongly polynomial max flow algorithm. $O(V^2E)$
Extremly fast **and works better** on biprtite graph. $O(\sqrt{VE})$

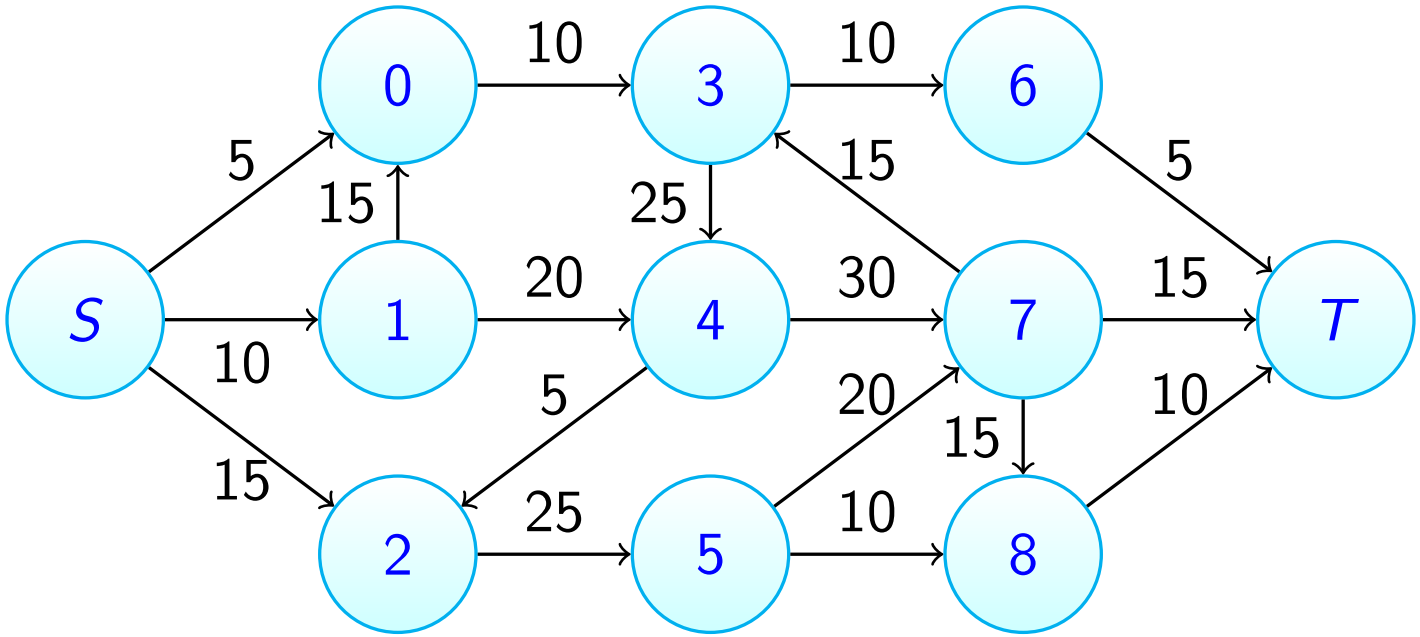
1. Construct a level graph by doing a BFS from source to label all the levels of the current flow graph.
2. If the sink was never reached while building the level graph, then stop and return max flow.
3. Using only valid edges in the level graph, do multiple DFSs from s->t until a blocking flow is reached, and sum over the bottleneck values of all the augmenting paths found to calculate the max flow.

MaxFlow: Dinic's

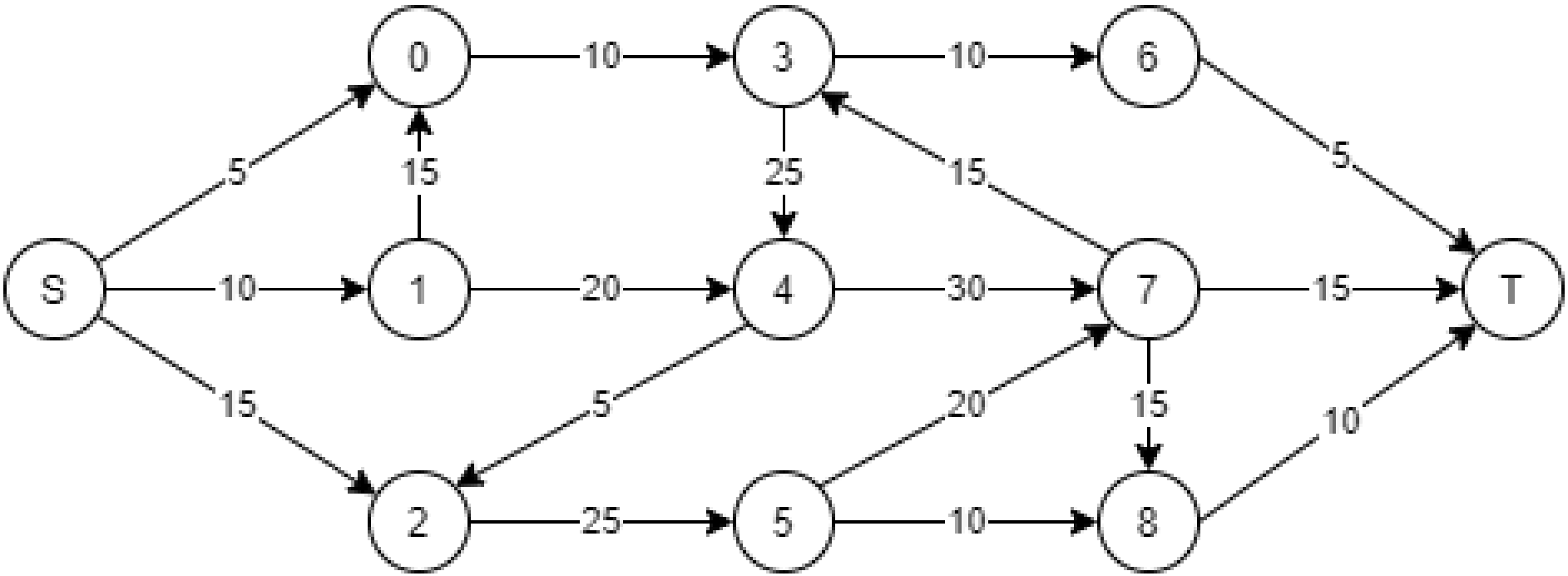
Strongly polynomial max flow algorithm. $O(V^2E)$
Extremly fast **and works better** on biprtite graph. $O(\sqrt{VE})$

1. Construct a level graph by doing a BFS from source to label all the levels of the current flow graph.
2. If the sink was never reached while building the level graph, then stop and return max flow.
3. Using only valid edges in the level graph, do multiple DFSs from s->t until a blocking flow is reached, and sum over the bottleneck values of all the augmenting paths found to calculate the max flow.

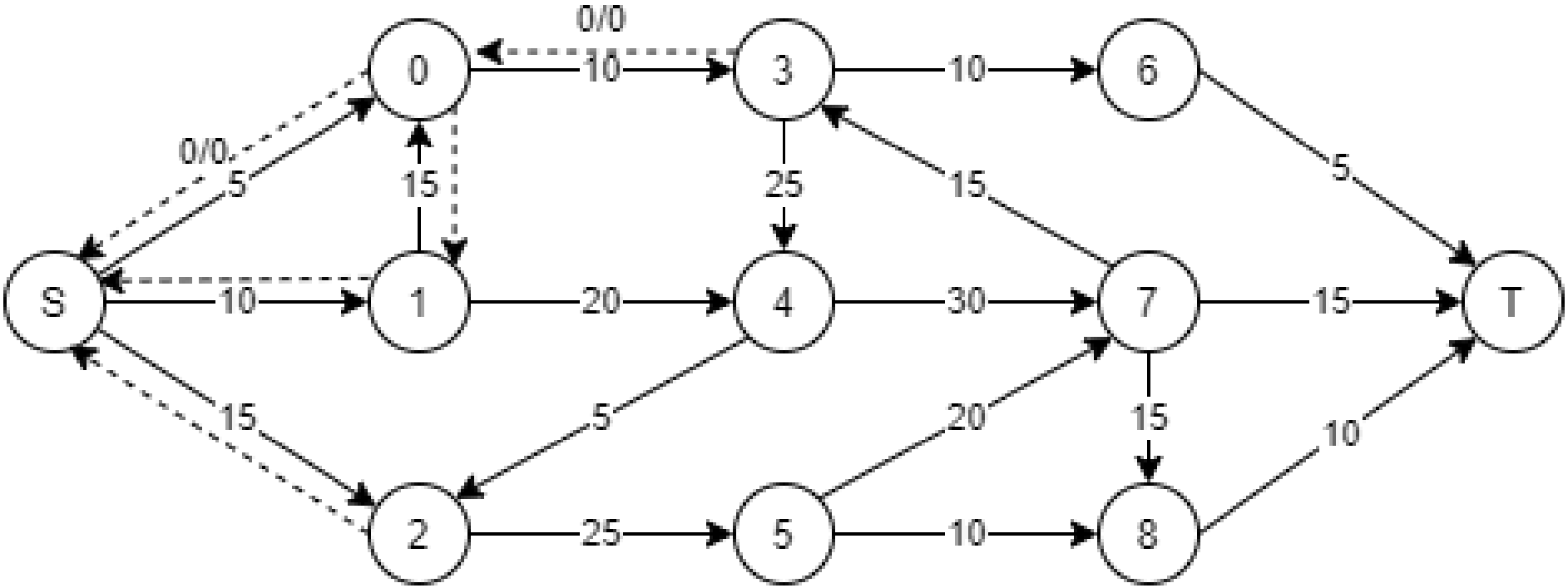
MaxFlow: Dinic's



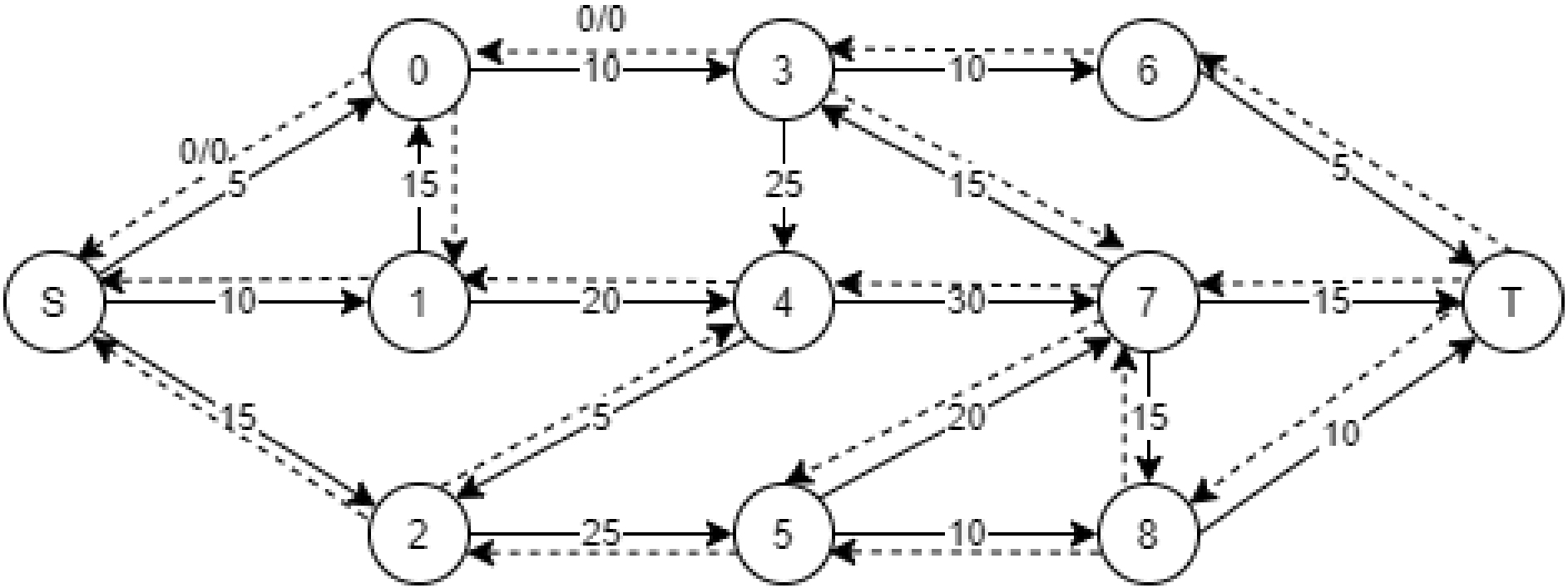
MaxFlow: Dinic's



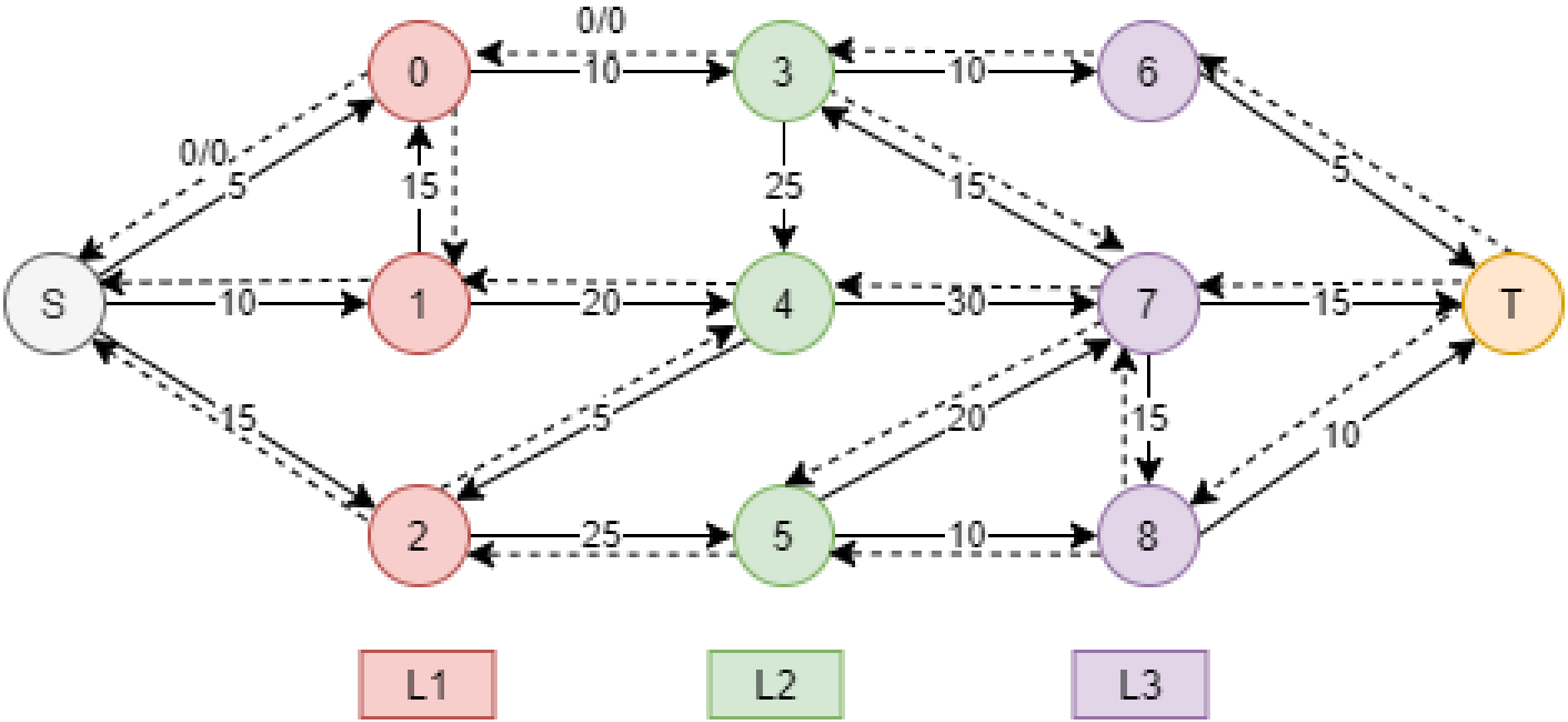
MaxFlow: Dinic's



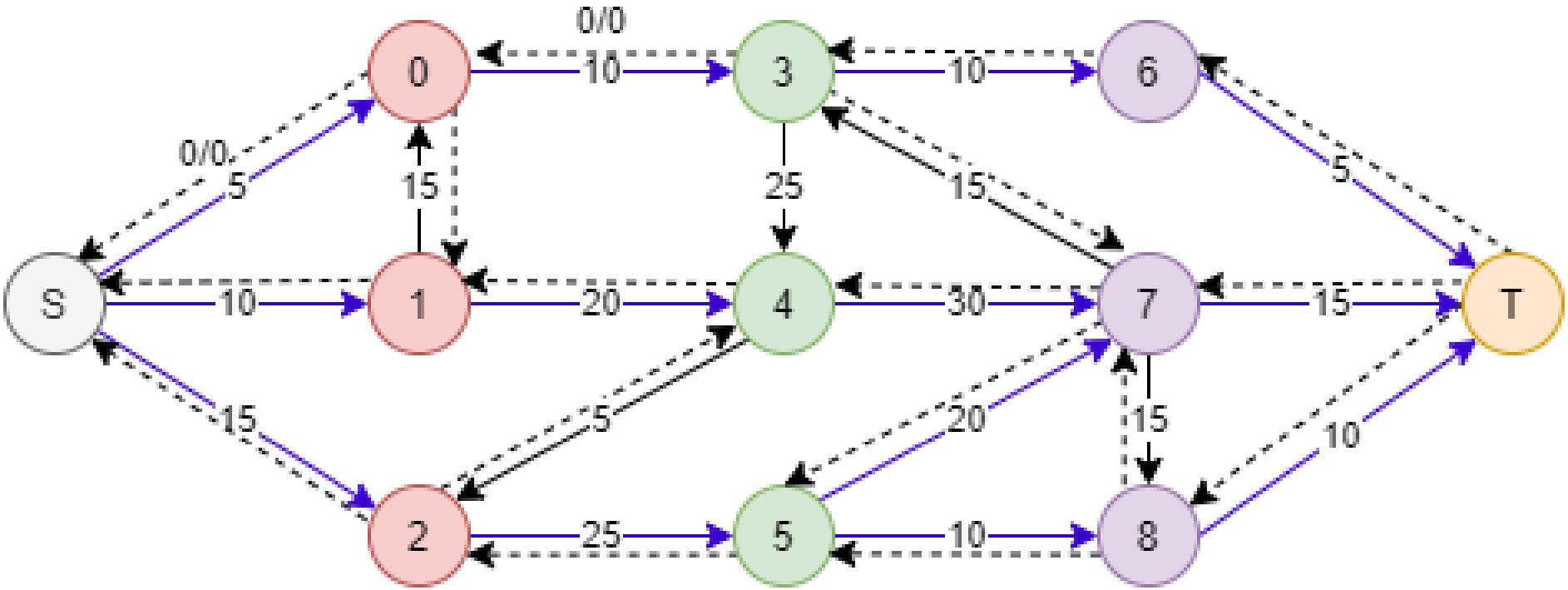
MaxFlow: Dinic's



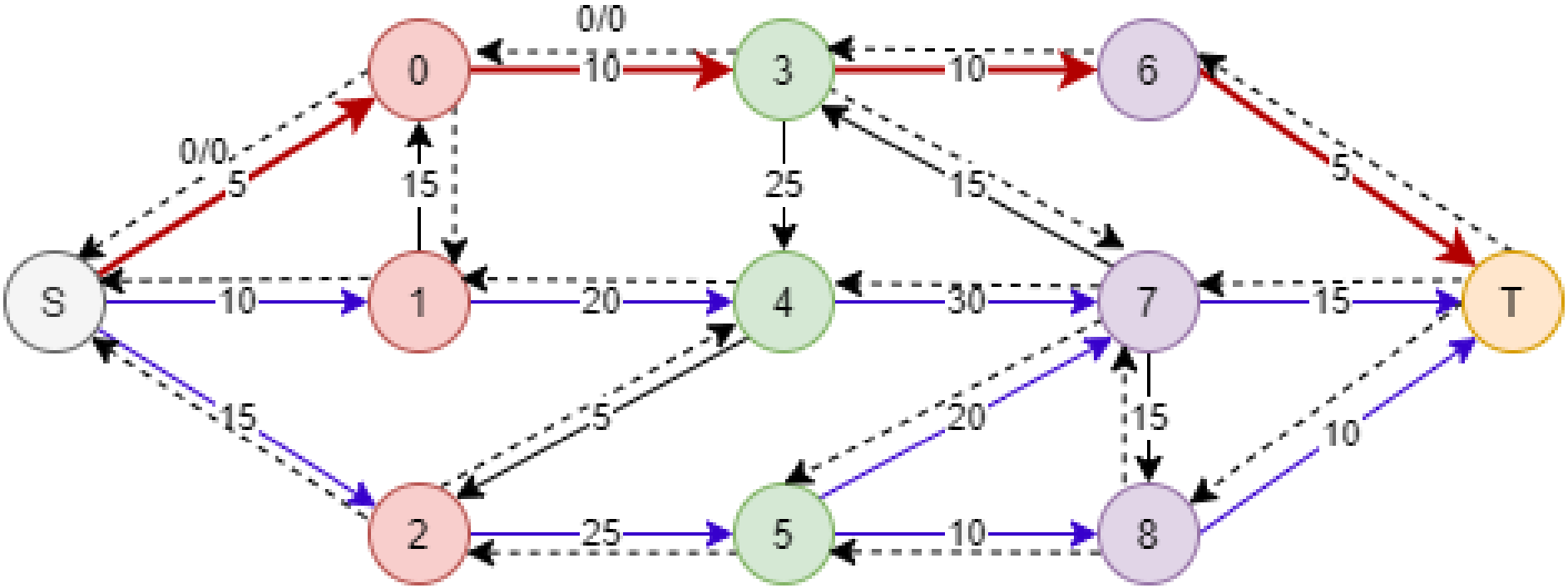
MaxFlow: Dinic's



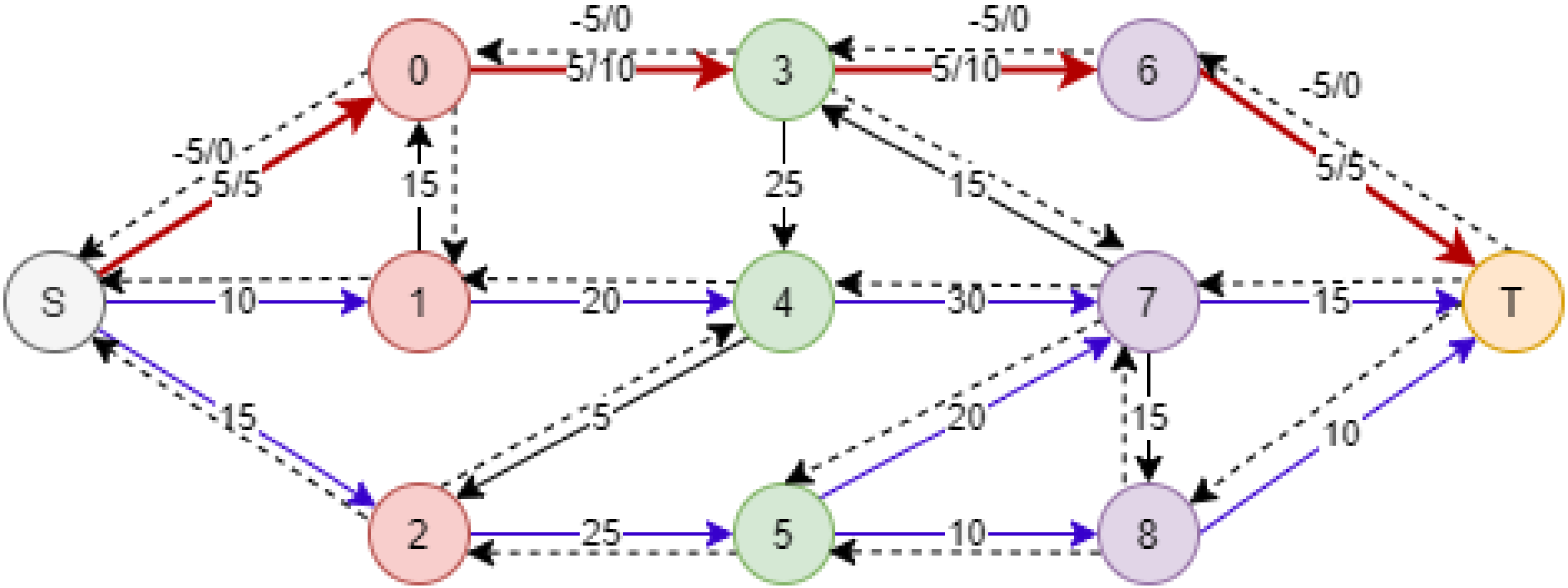
MaxFlow: Dinic's



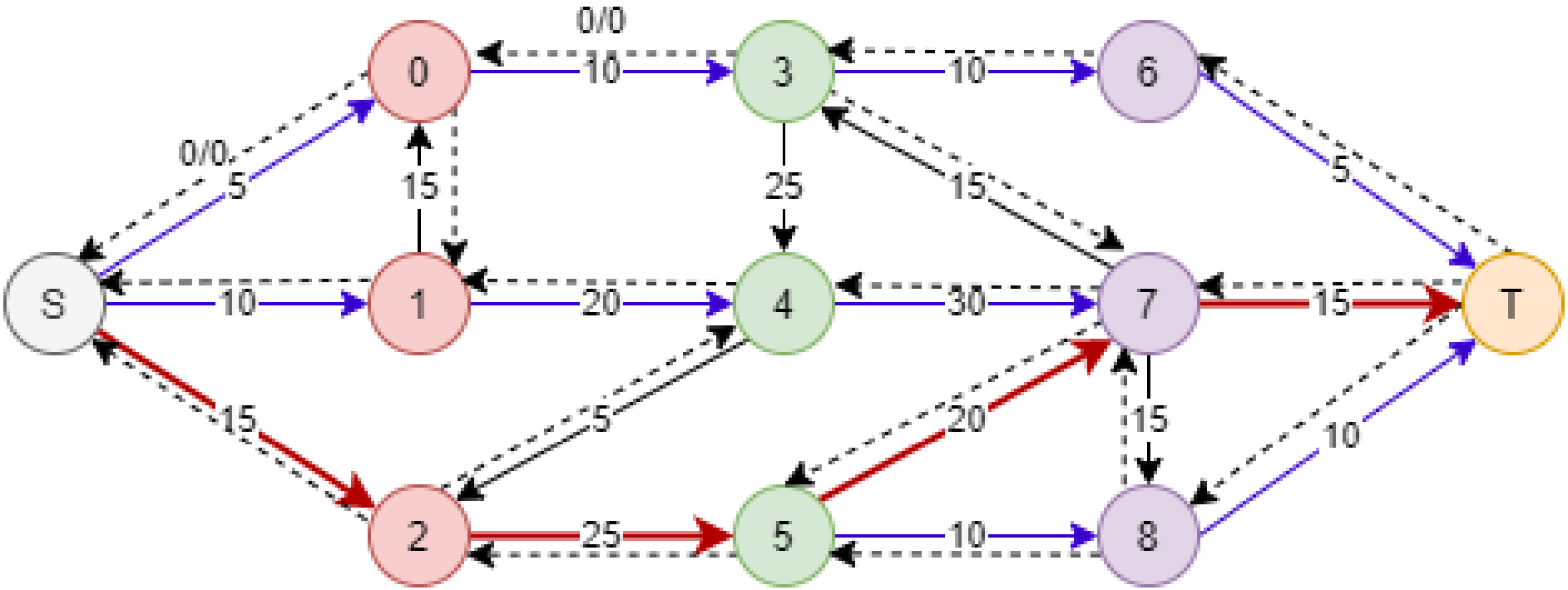
MaxFlow: Dinic's



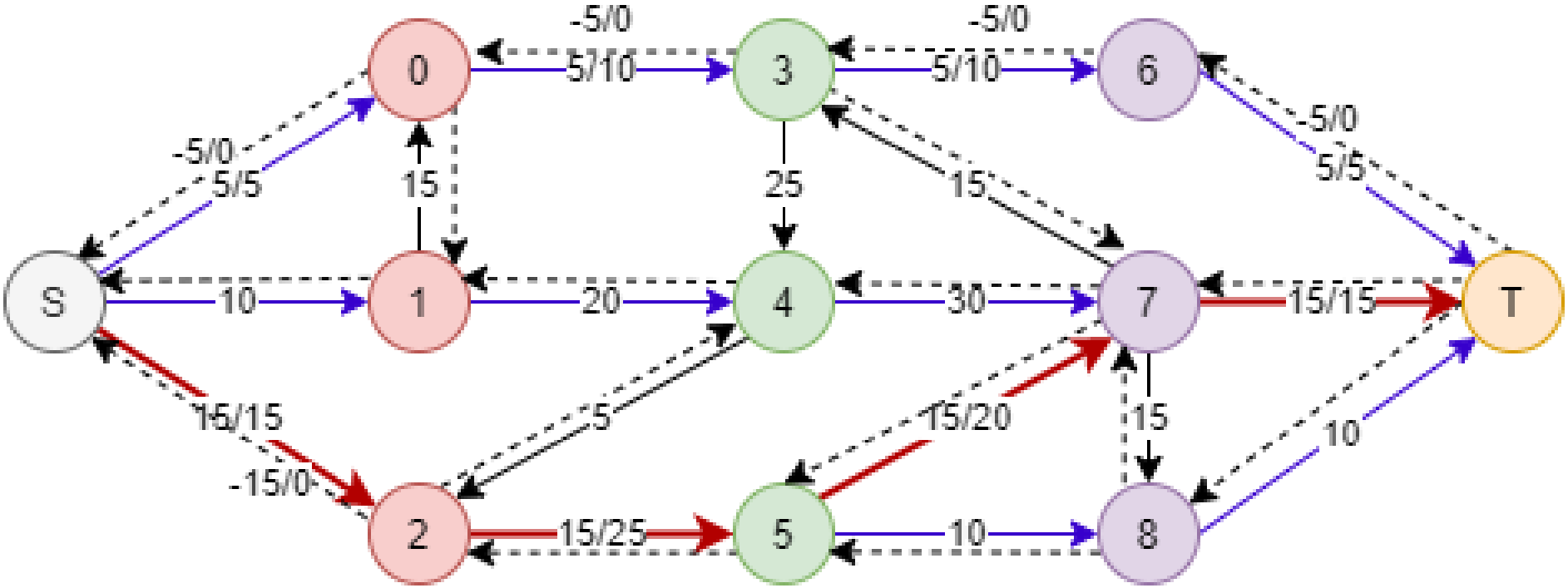
MaxFlow: Dinic's



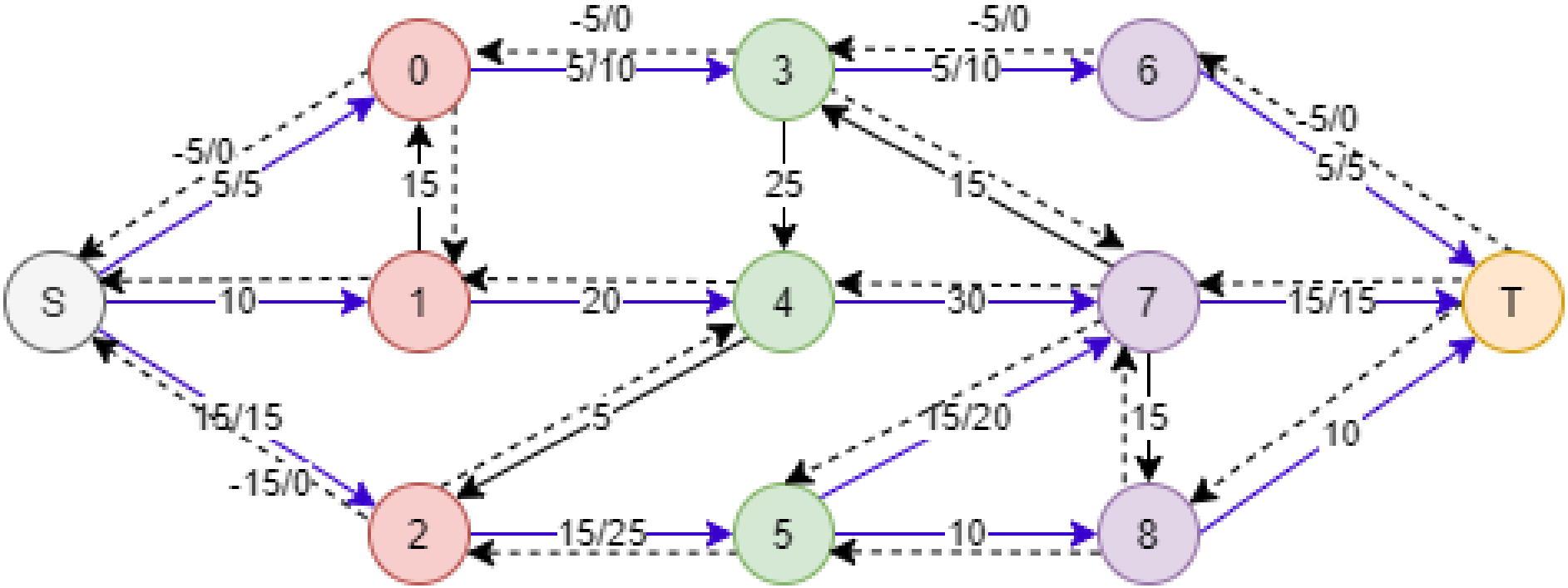
MaxFlow: Dinic's



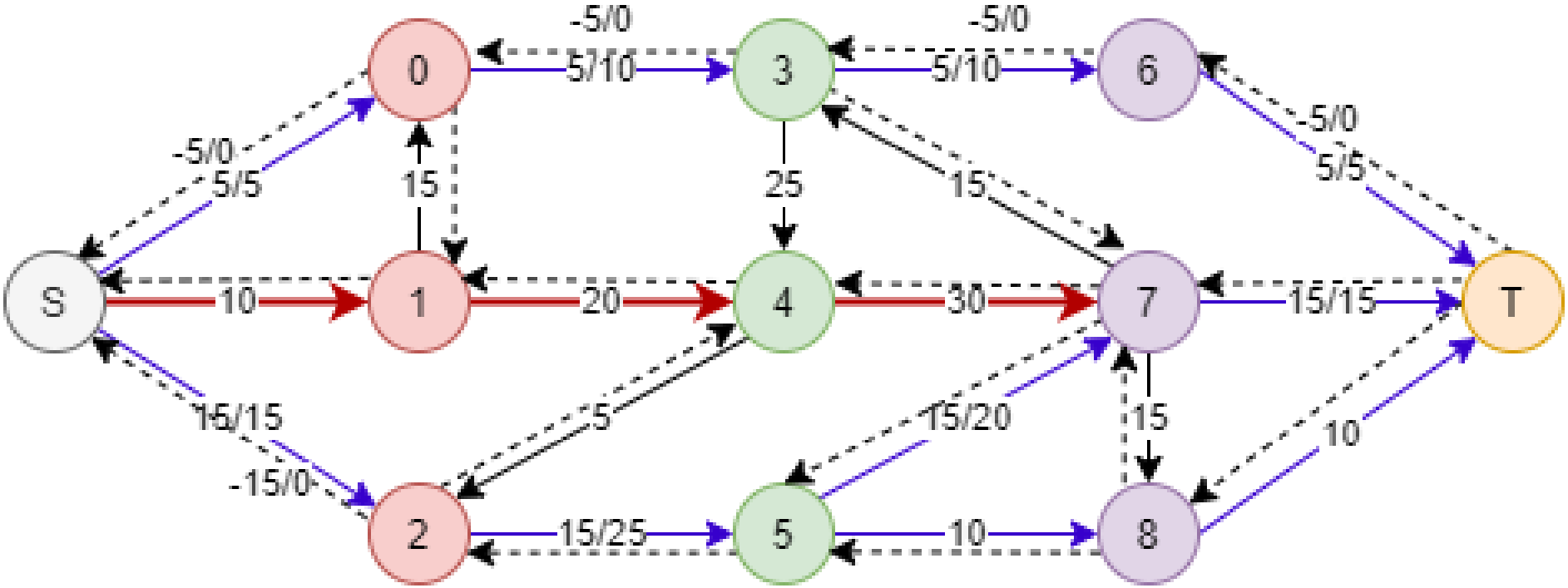
MaxFlow: Dinic's



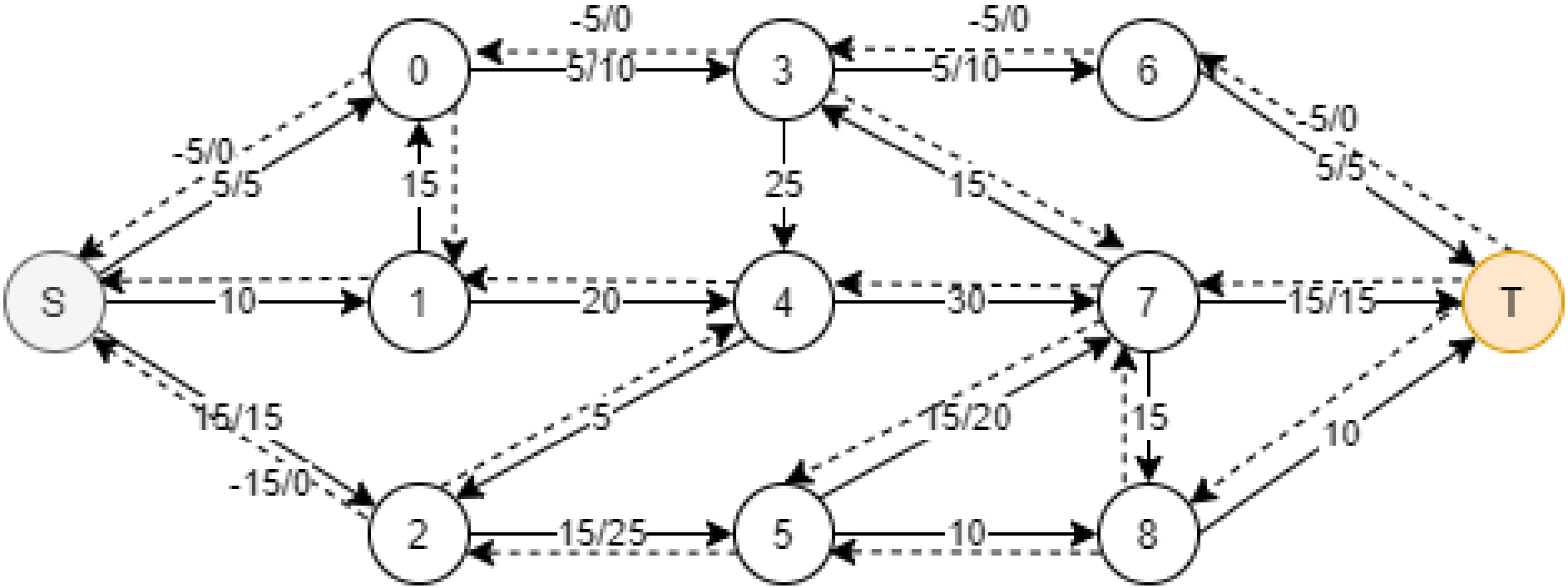
MaxFlow: Dinic's



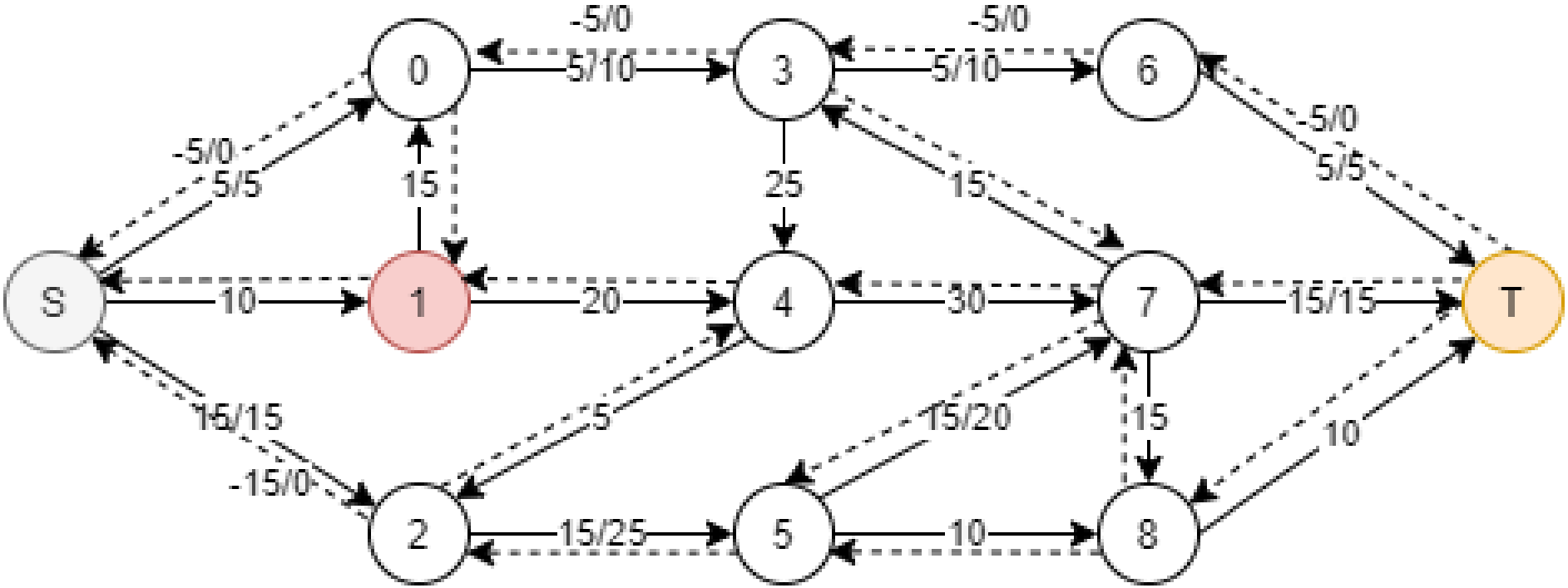
MaxFlow: Dinic's



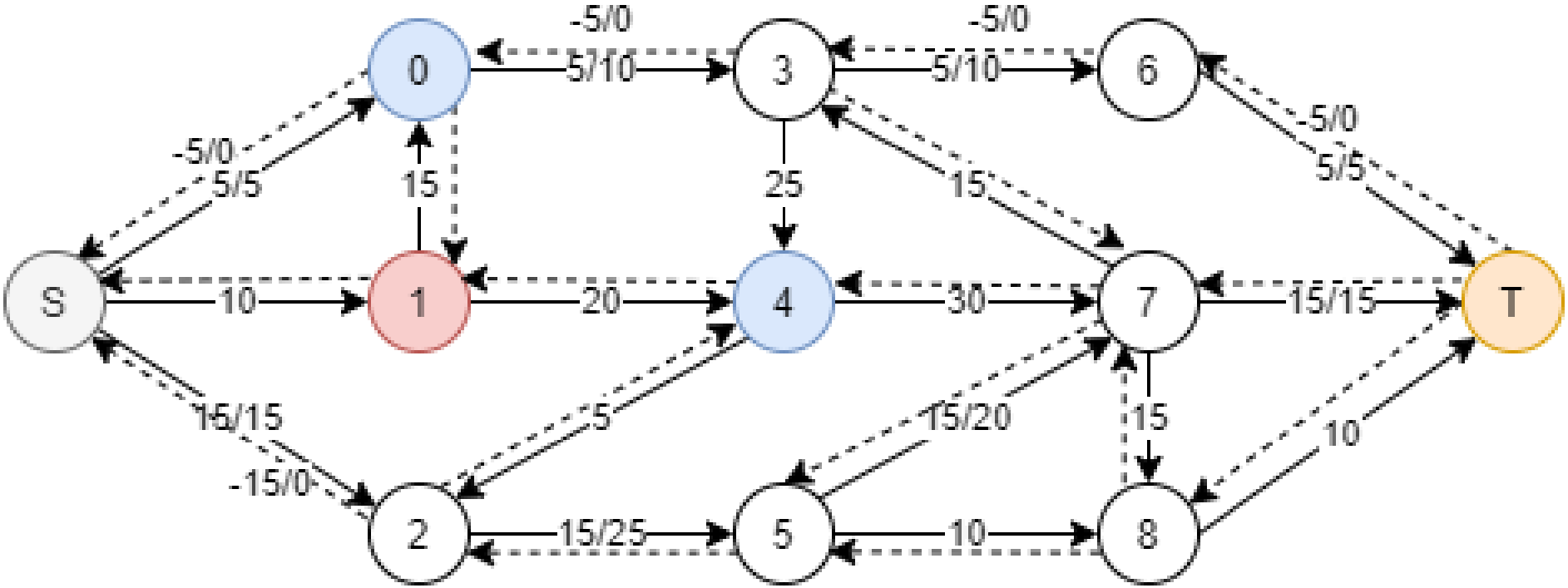
MaxFlow: Dinic's



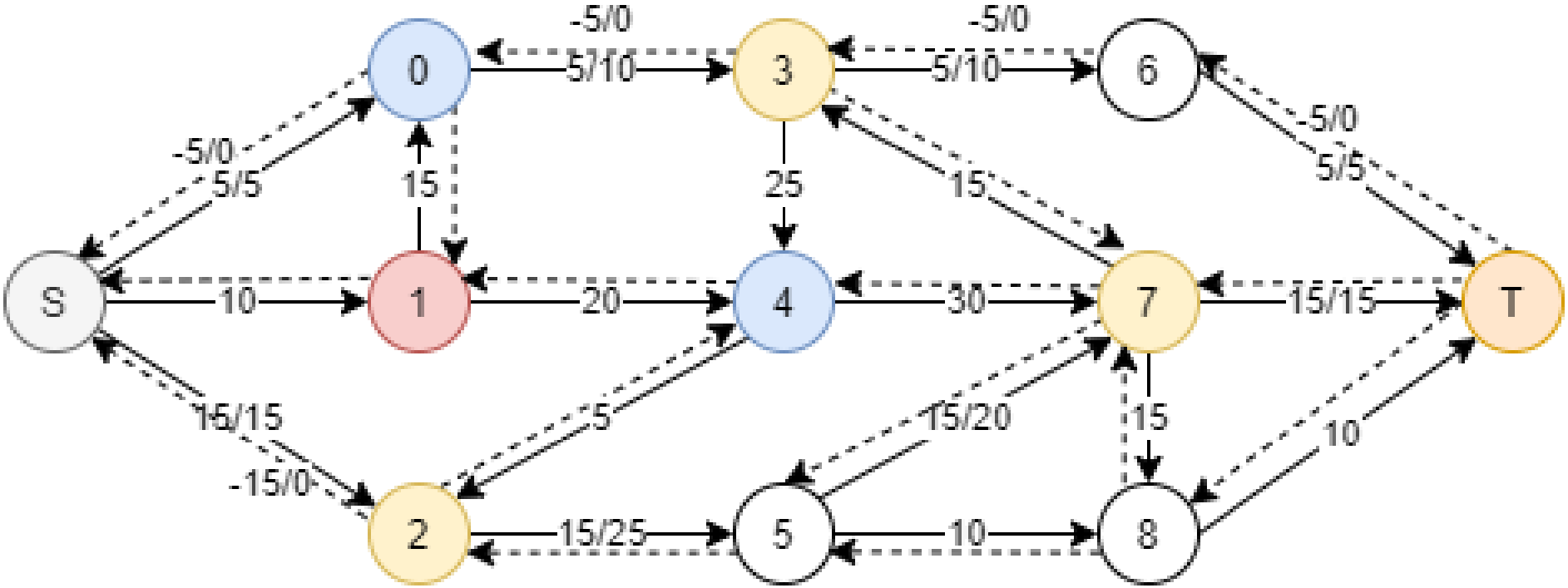
MaxFlow: Dinic's



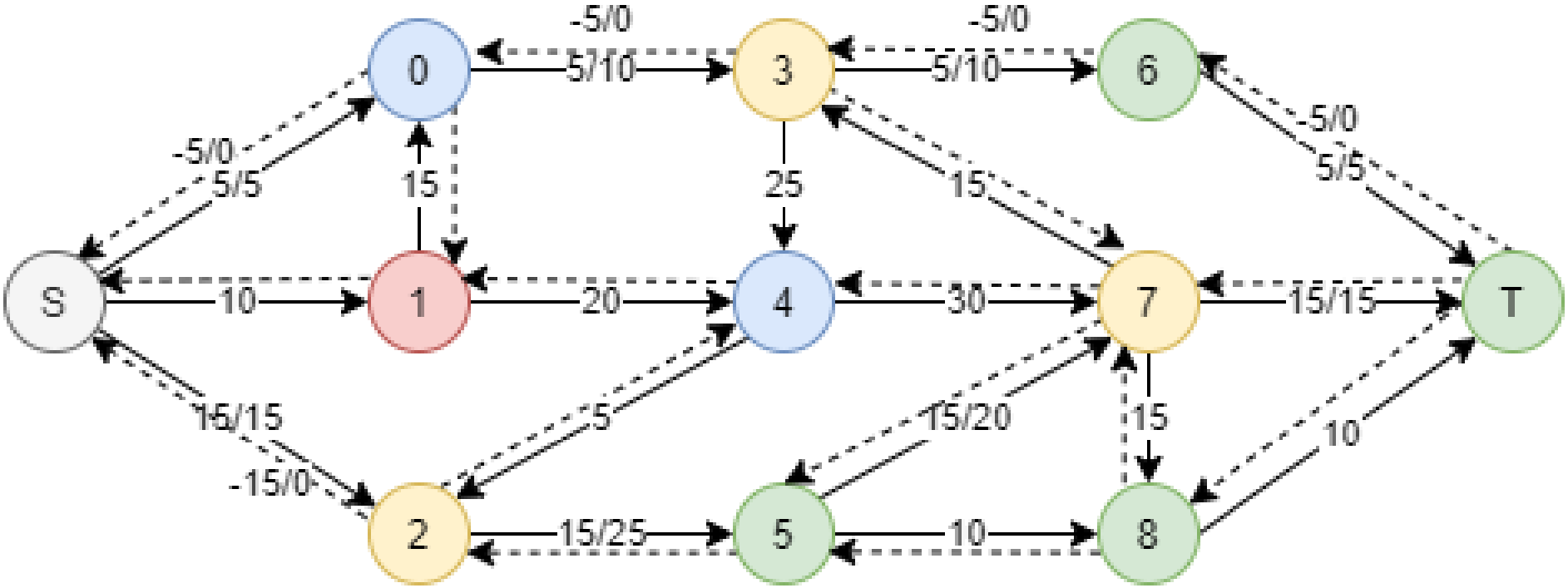
MaxFlow: Dinic's



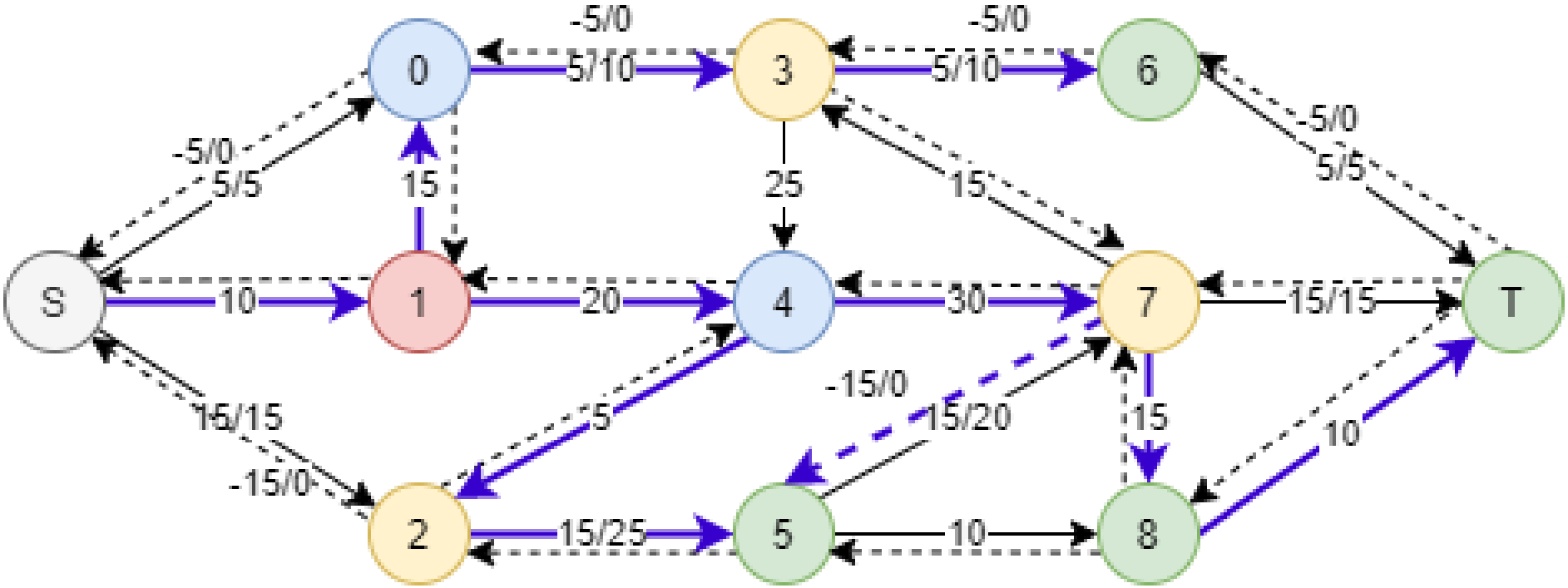
MaxFlow: Dinic's



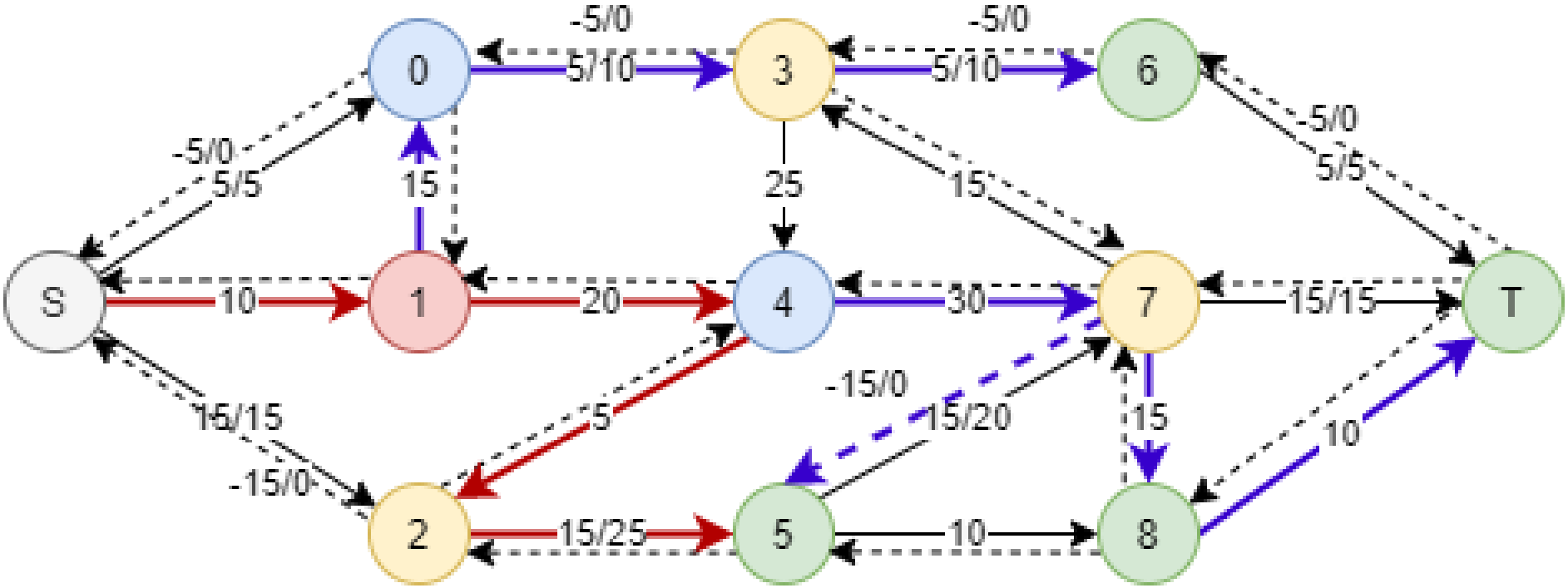
MaxFlow: Dinic's



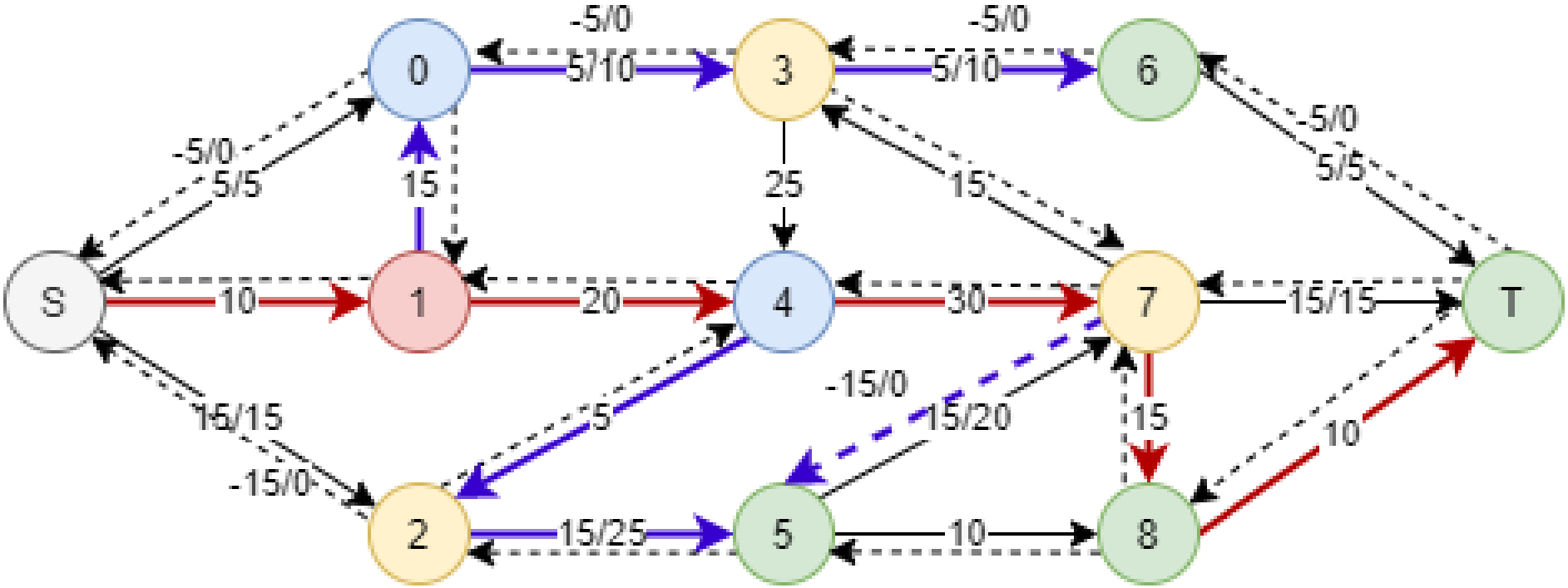
MaxFlow: Dinic's



MaxFlow: Dinic's



MaxFlow: Dinic's



MaxFlow: Exercise