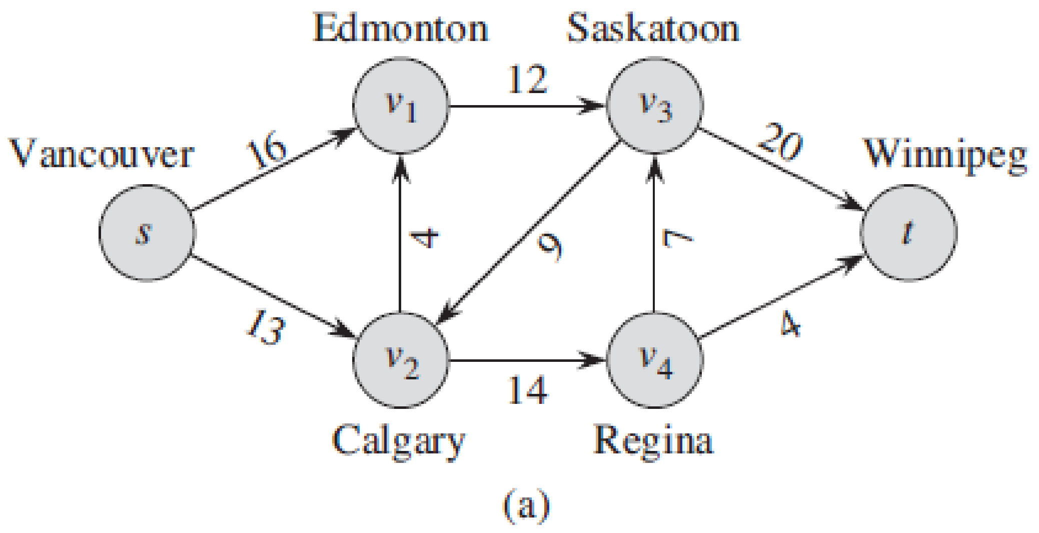
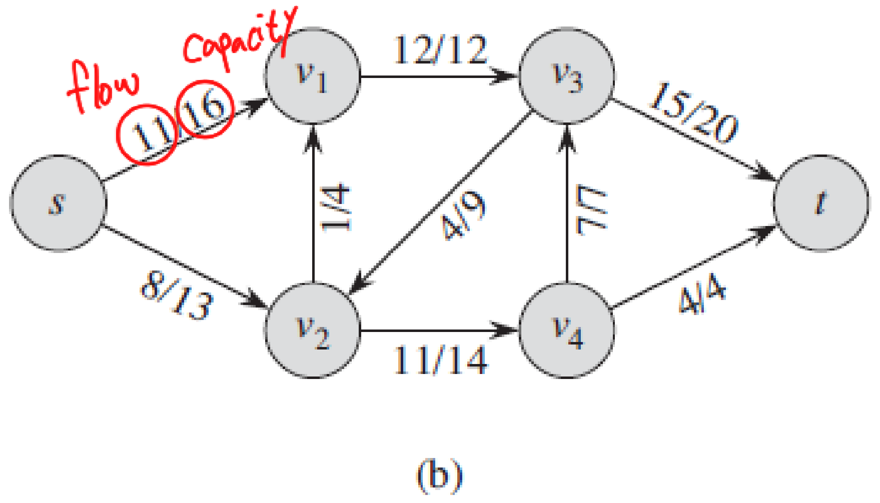
Maximum Flow

Flow Networks

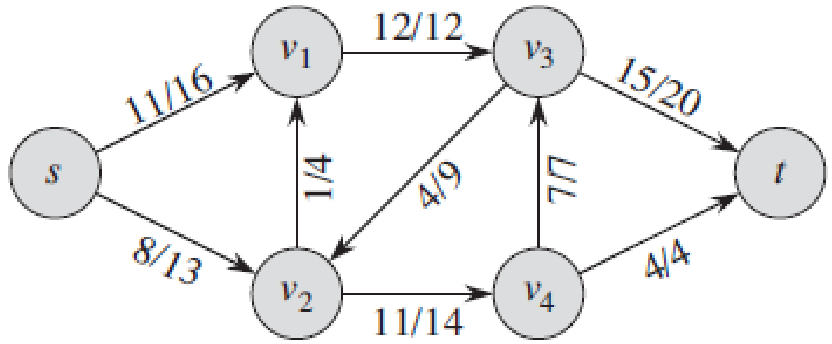
- Flow network G=(V,E)
  - Directed graph with nonnegative capacity for every edge
  - Two special vertices: Source(*s*) and Sink(*t*)





Flow Networks

- Capacity constraints
  - $\forall (u, v) \in E: f(u, v) \leq c(u, v)$
  - The flow along the edge can not exceed its capacity
- Skew symmetry
  - $\forall (u, v) \in E: f(u, v) = -f(v, u)$
  - The net flow from u to v must be the opposite of the net flow from v to u
- Flow conservation
  - The net flow to a node is zero, except for the source and sink
- Value(f)
  - $\sum_{(s,u)\in E} f(s, u) = \sum_{(v,t)\in E} f(v, t)$



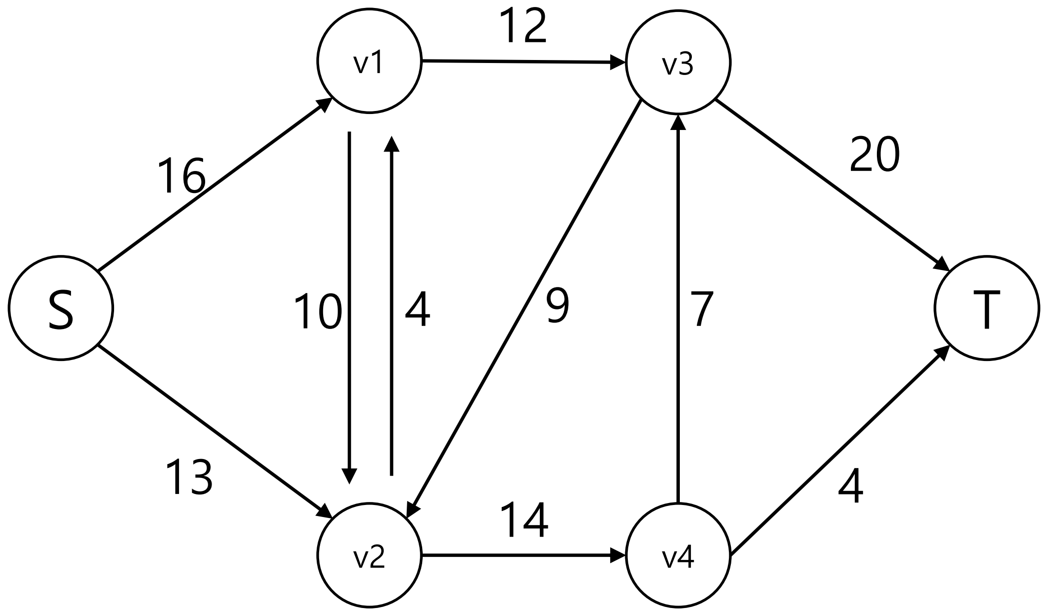
Ford-Fulkerson Algorithm

- Developed by L.R.Ford, Jr. and D.R.Fulkerson in 1956

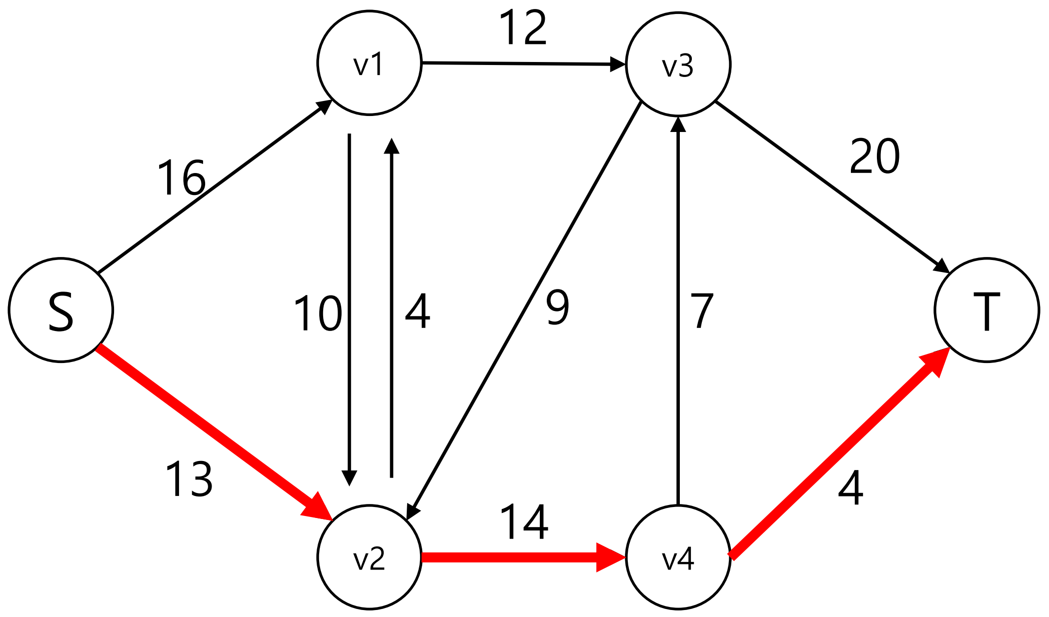
```
function: FordFulkerson(Graph G,Node S,Node T):
    Initialise flow in all edges to 0
    while (there exists an augmenting path(P) between S and T in residual network graph):
        Augment flow between S to T along the path P
        Update residual network graph
    return
```

- Residual network graph indicates how much more flow is allowed in each edge in the network graph.
- Augmenting path: a simple path which do not include any cycles and that pass only through positive weighted edges.

Ford-Fulkerson Algorithm

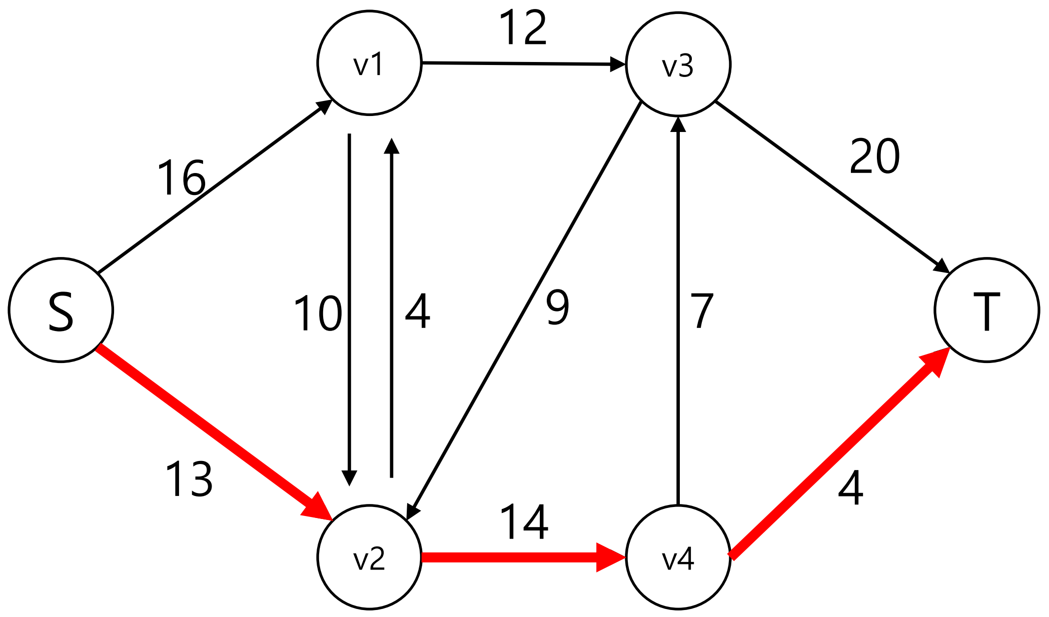


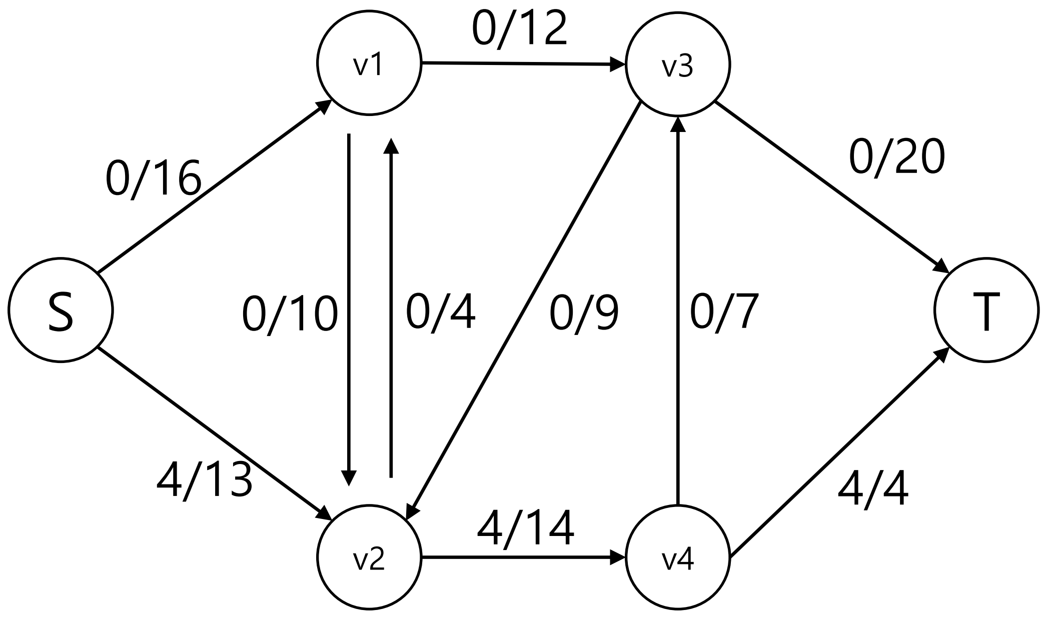
Ford-Fulkerson Algorithm



Augmenting path

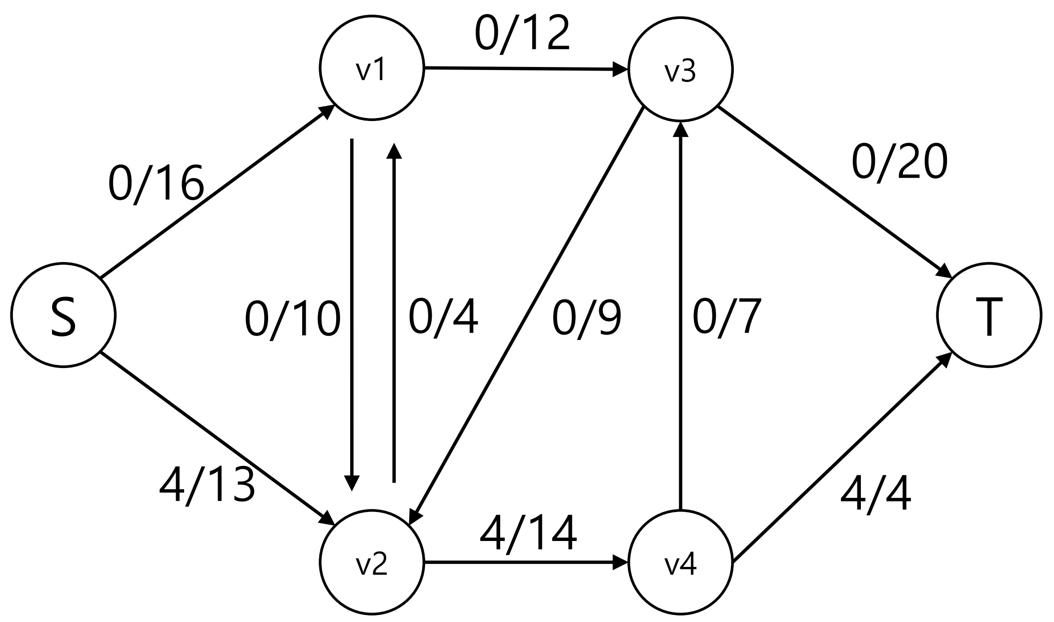
Ford-Fulkerson Algorithm





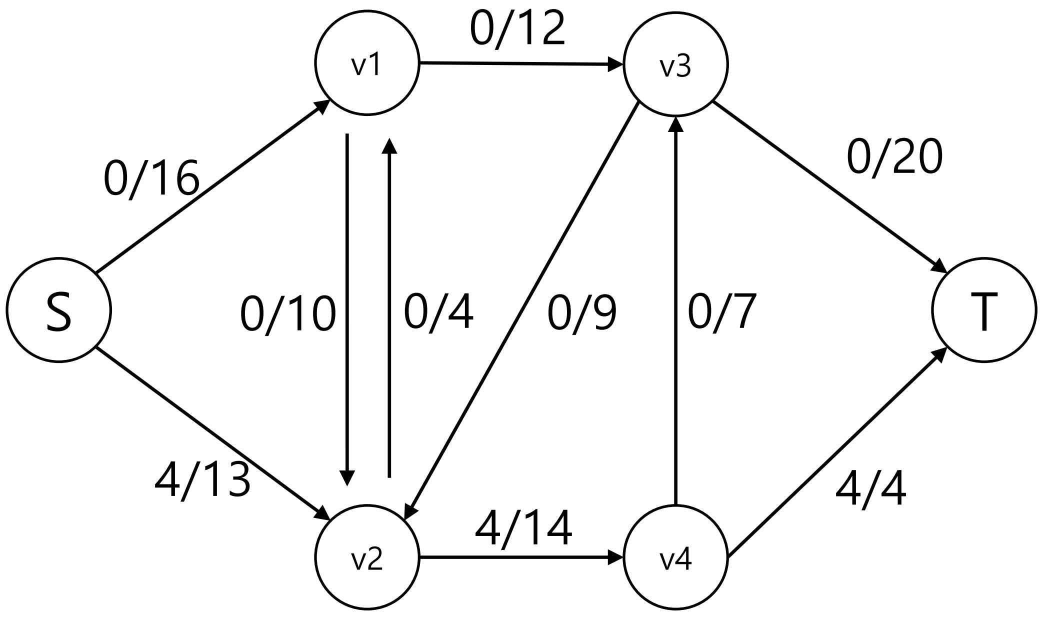
Flow network

Ford-Fulkerson Algorithm

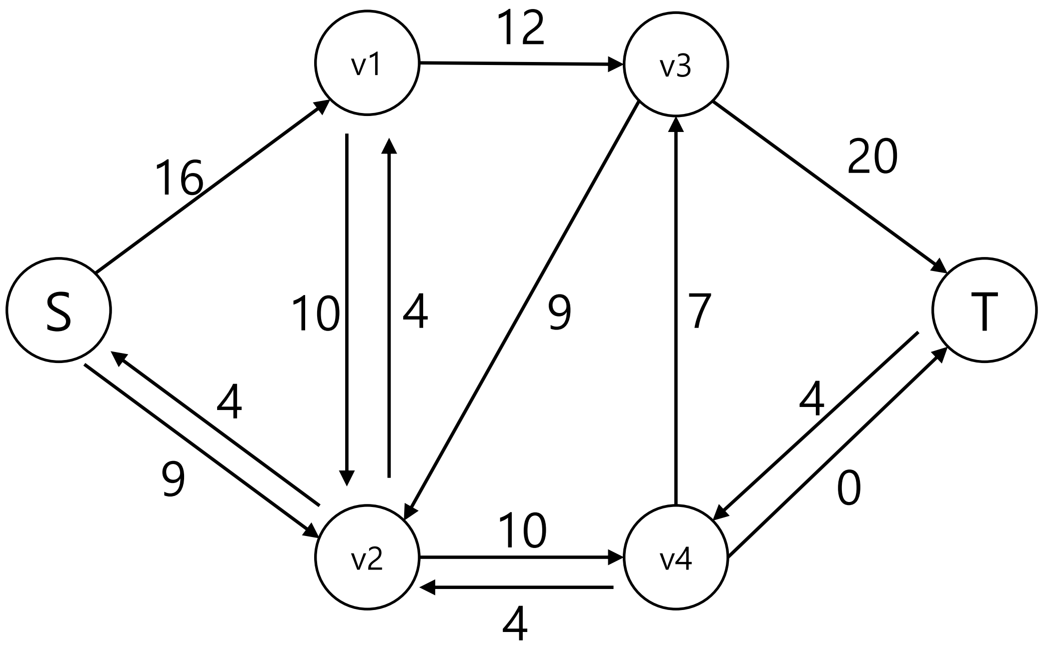


Flow network

Ford-Fulkerson Algorithm

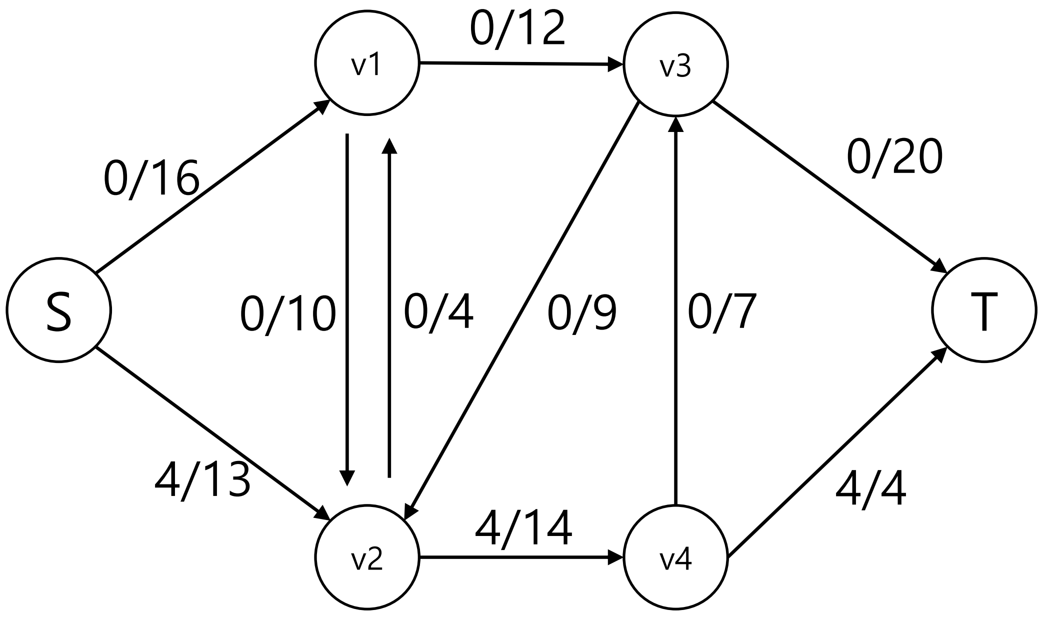


Flow network



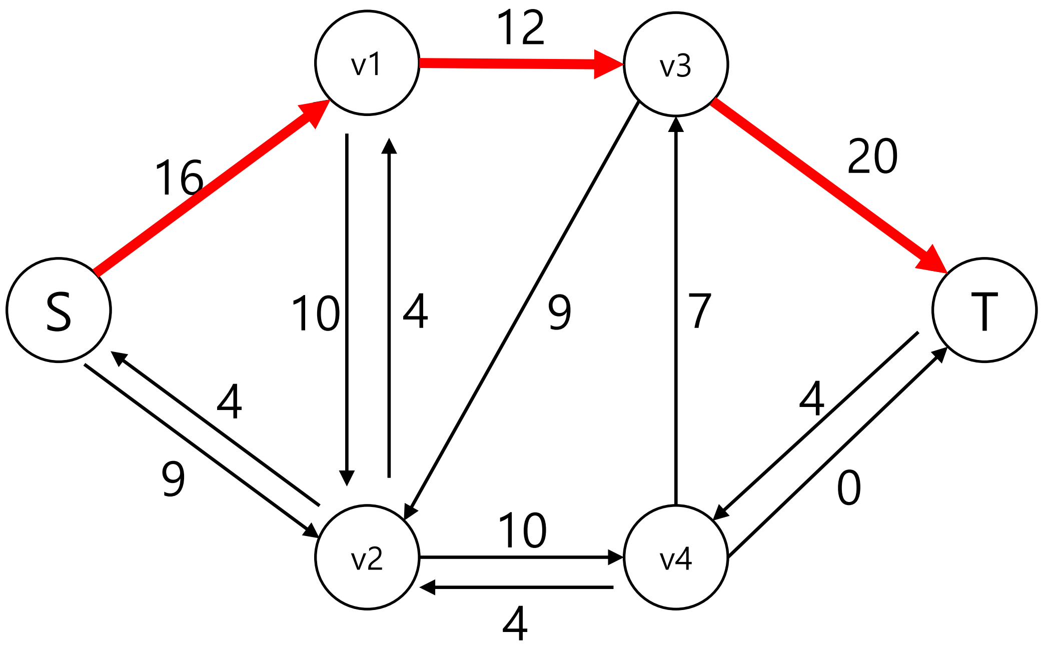
Residual Graph

Ford-Fulkerson Algorithm



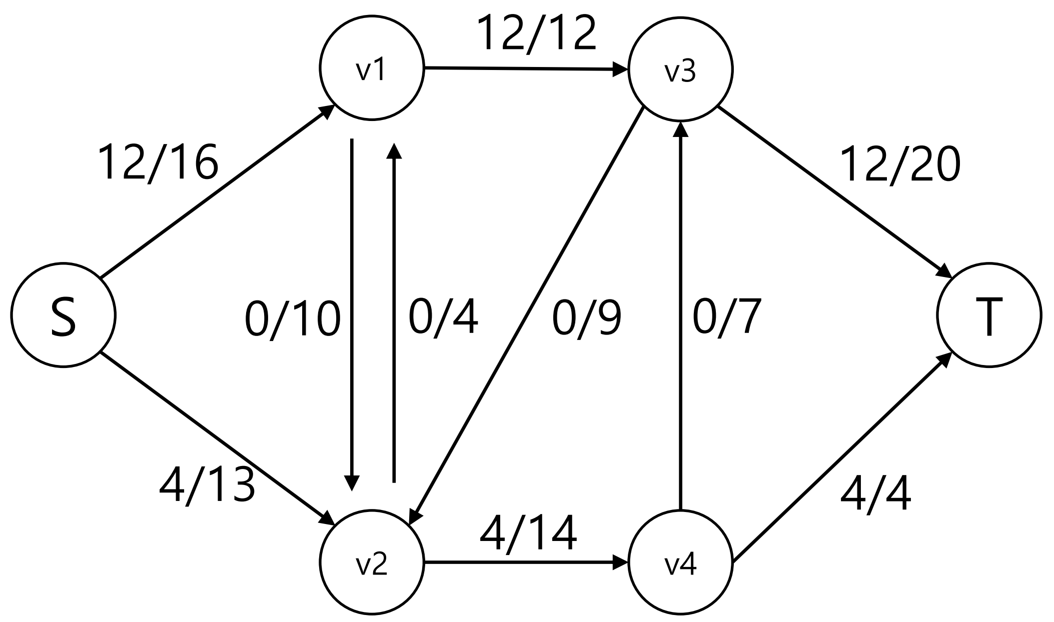
Flow network

Augmenting path: flow=12



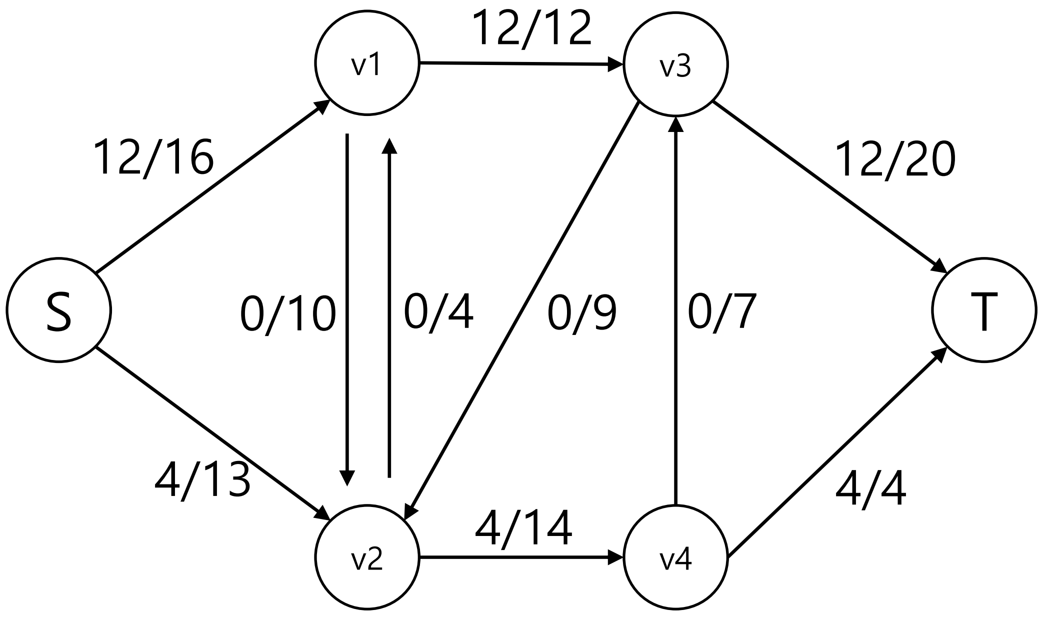
Residual Graph

Ford-Fulkerson Algorithm

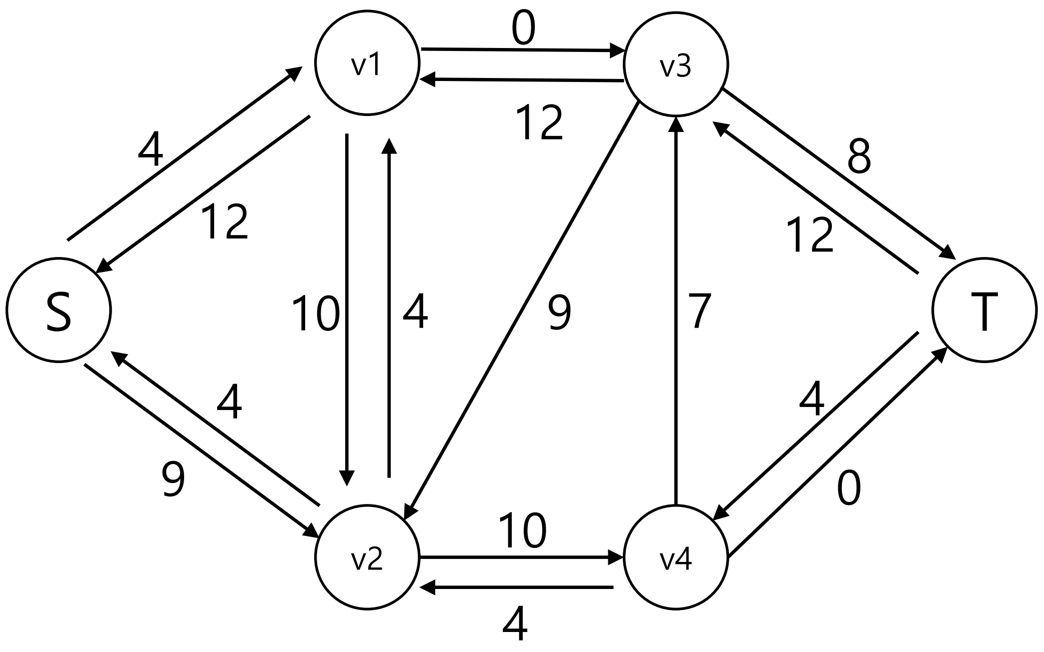


Flow network

Ford-Fulkerson Algorithm

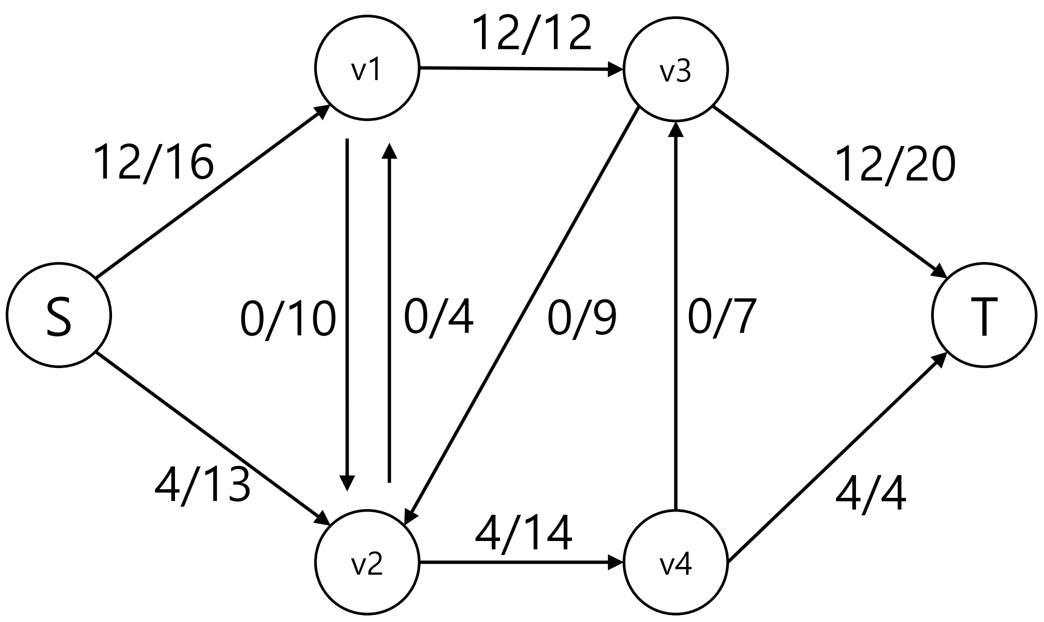


Flow network



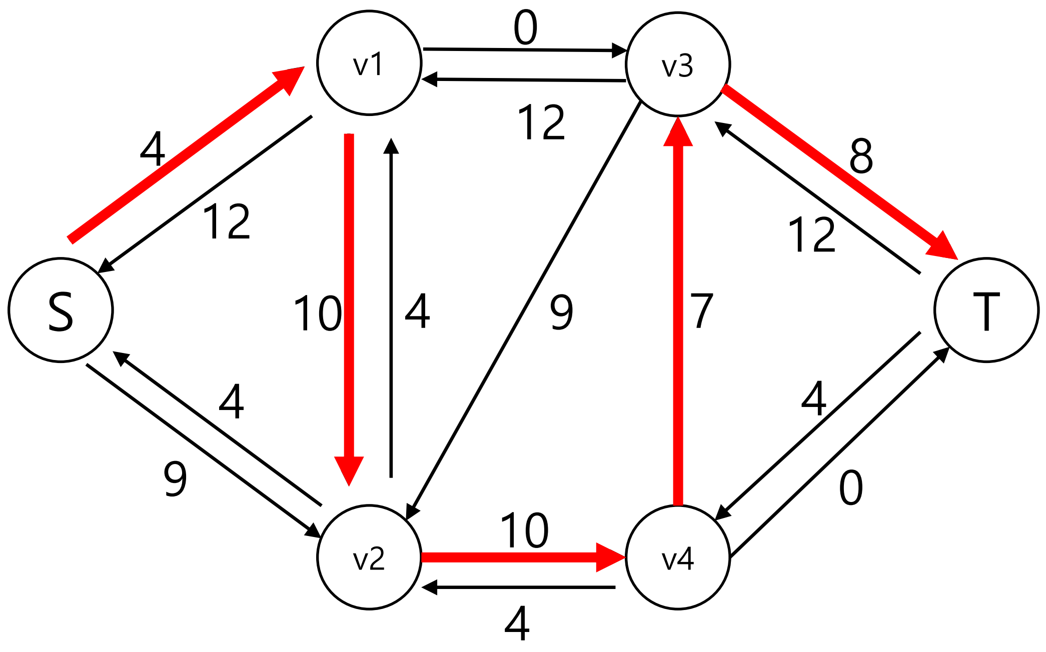
Residual Graph

Ford-Fulkerson Algorithm



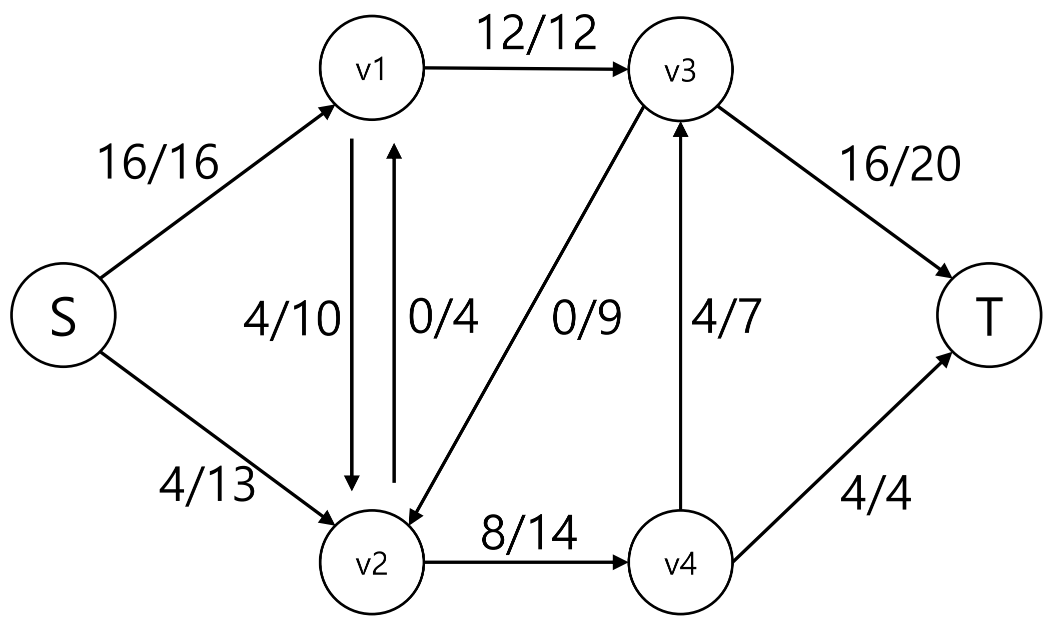
Flow network

Augmenting path: flow=4



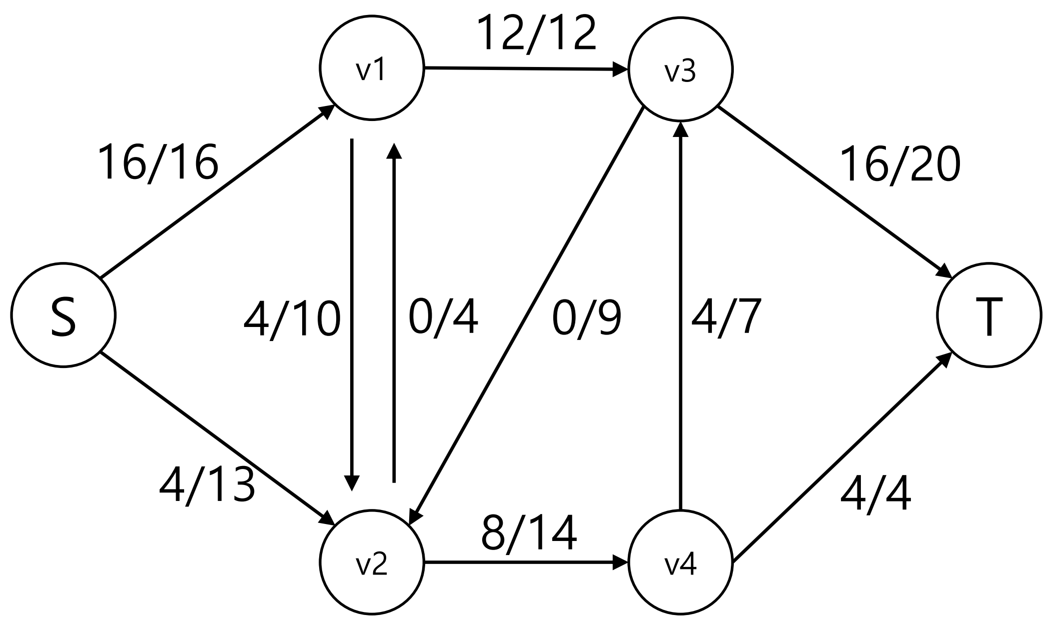
Residual Graph

Ford-Fulkerson Algorithm

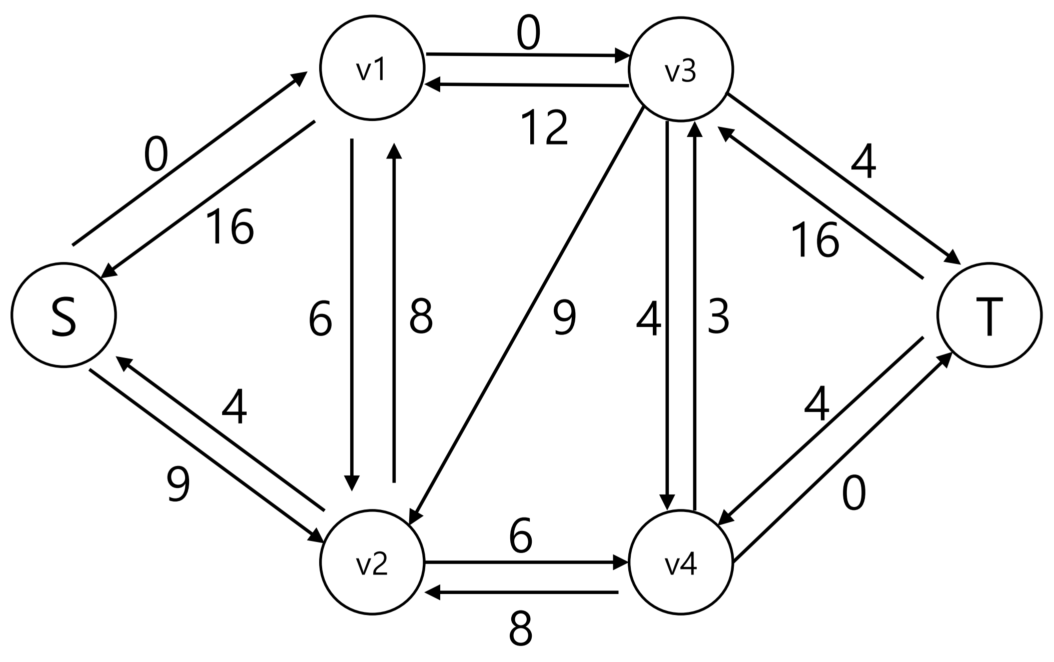


Flow network

Ford-Fulkerson Algorithm

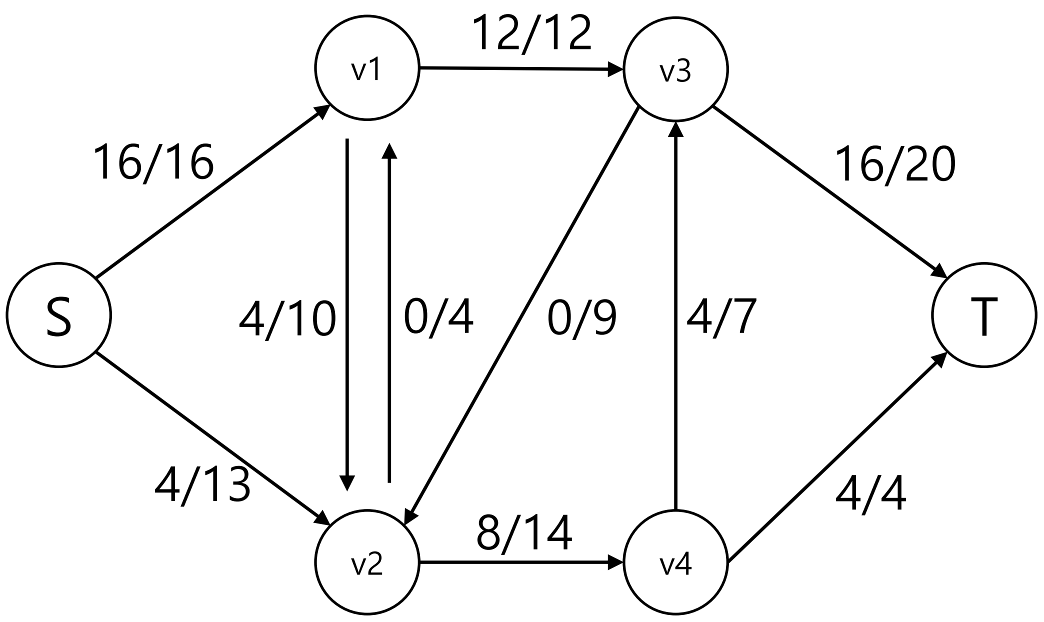


Flow network



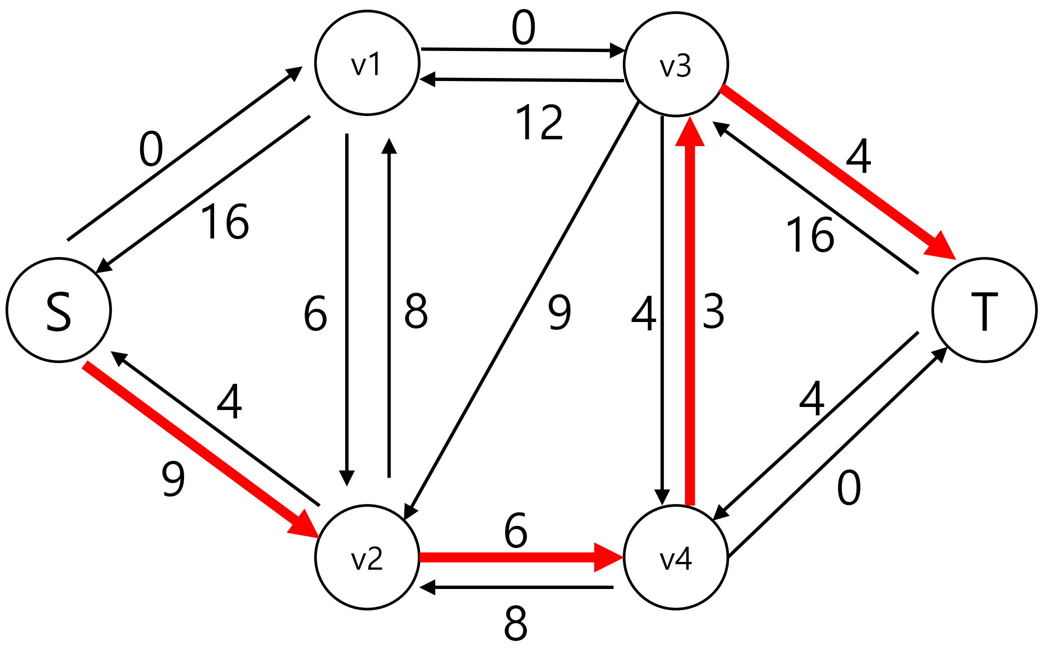
Residual Graph

Ford-Fulkerson Algorithm



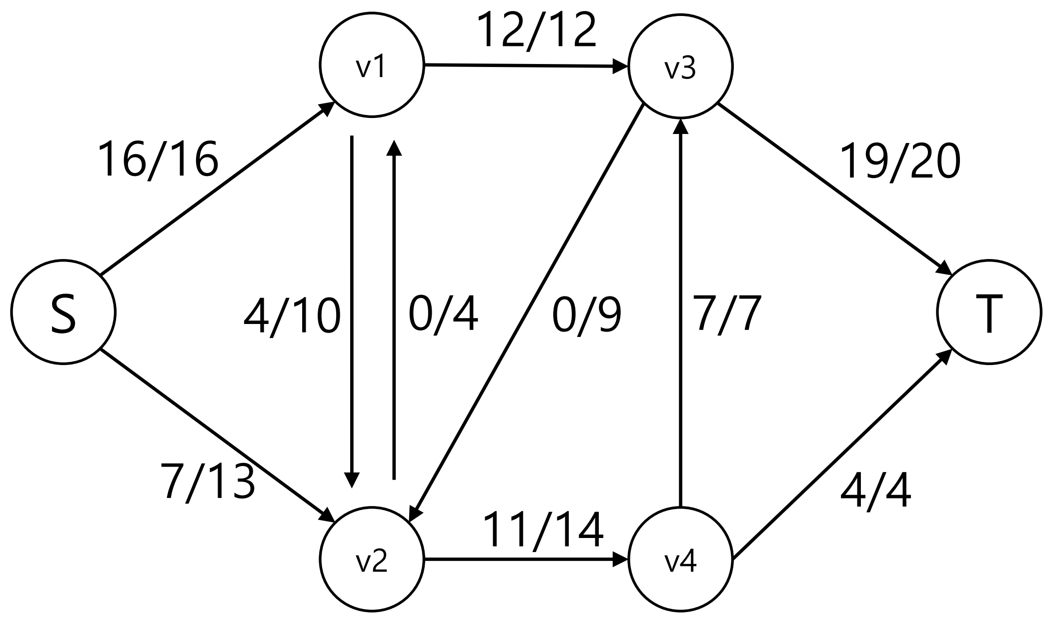
Flow network

Augmenting path: flow=3



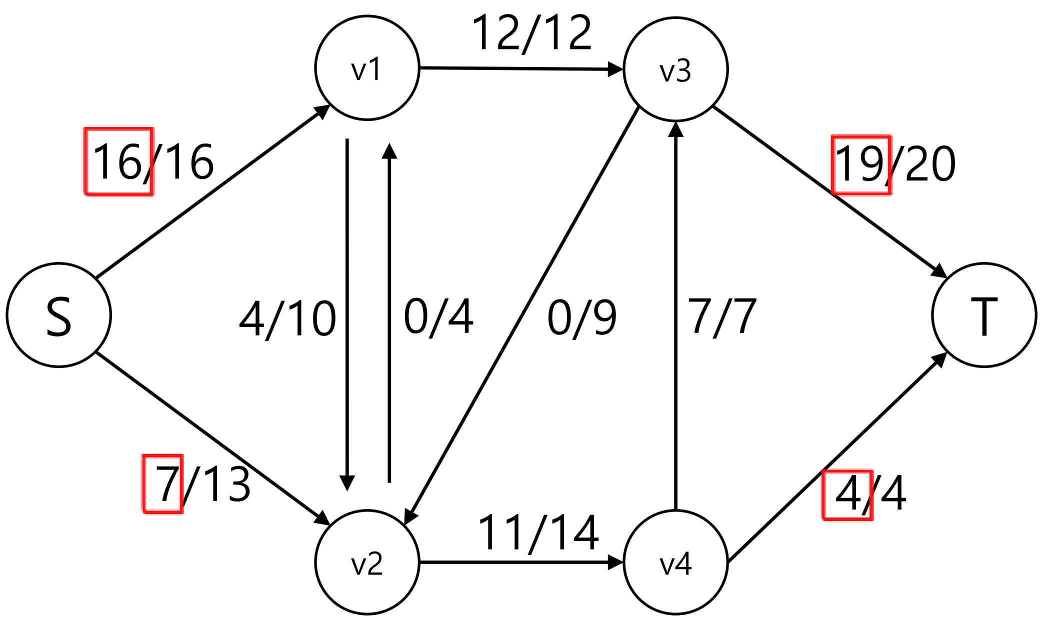
Residual Graph

Ford-Fulkerson Algorithm



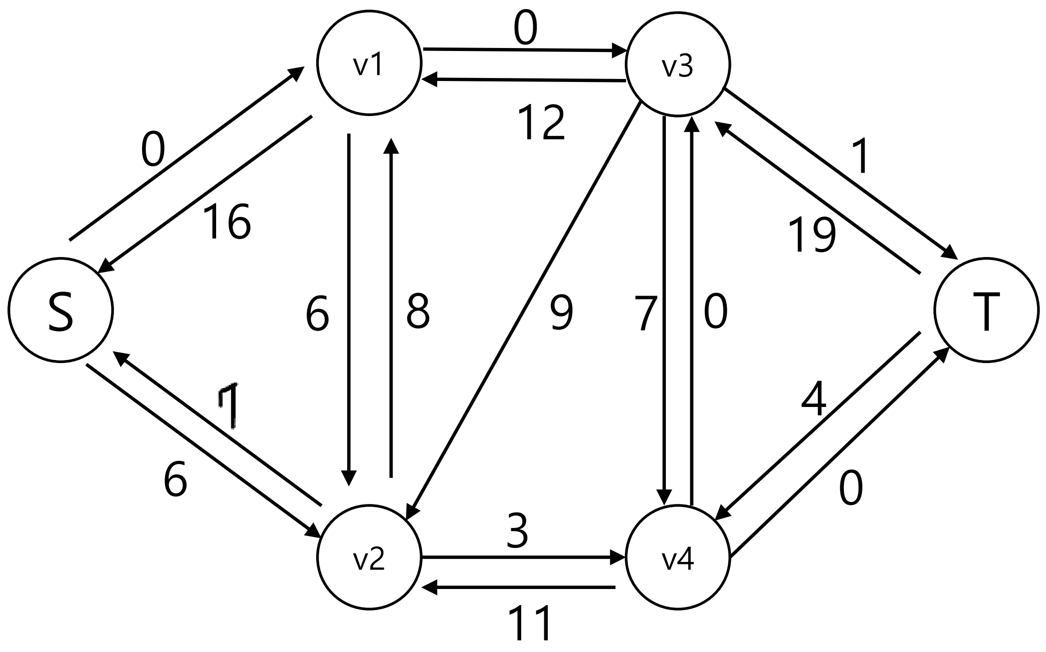
Flow network

Ford-Fulkerson Algorithm



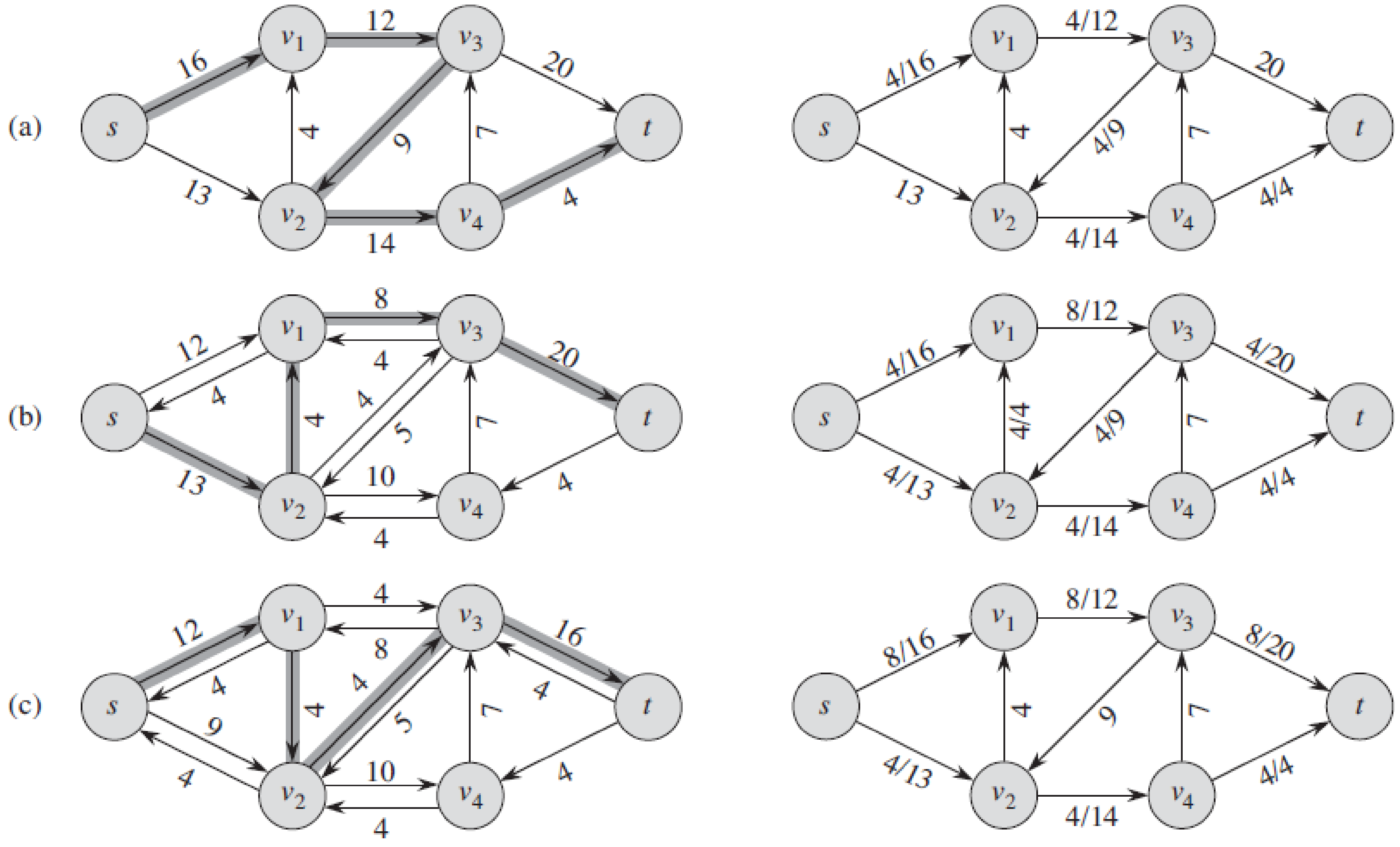
Flow network

No more augmenting path



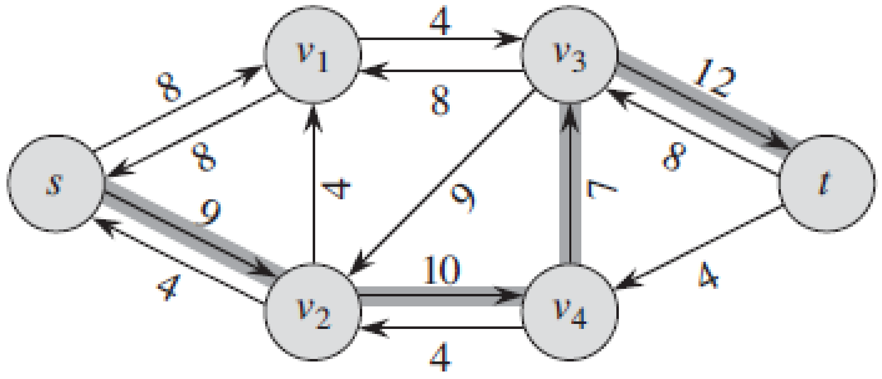
Max flow= 23

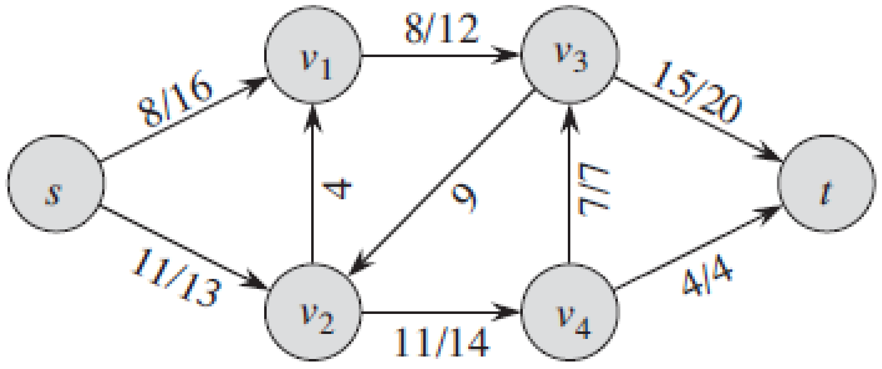
Ford-Fulkerson Algorithm : Augmenting path를 찾는 또 다른 순서



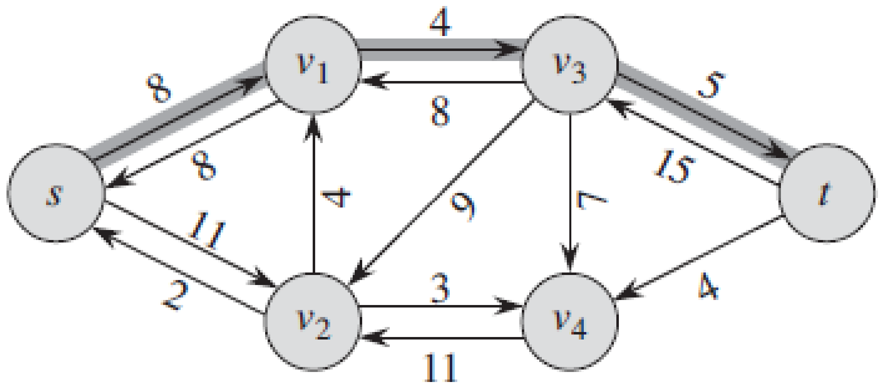
Ford-Fulkerson Algorithm : Augmenting path를 찾는 또 다른 순서

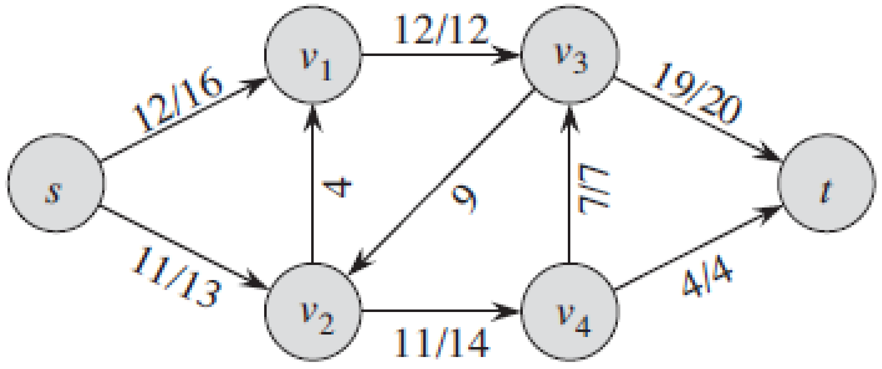
(d)



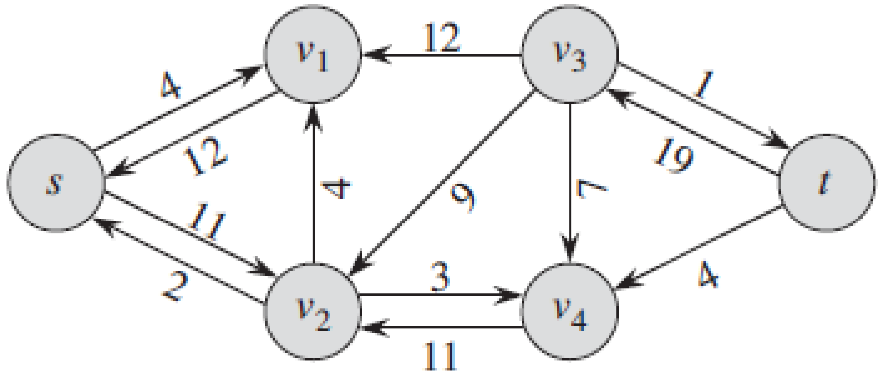


(e)



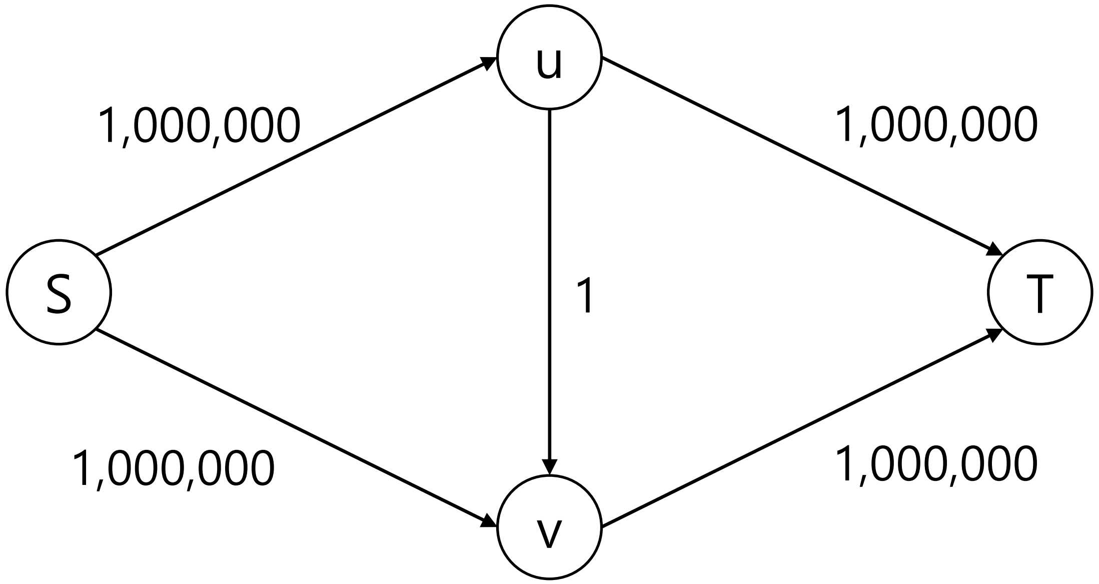


(f)



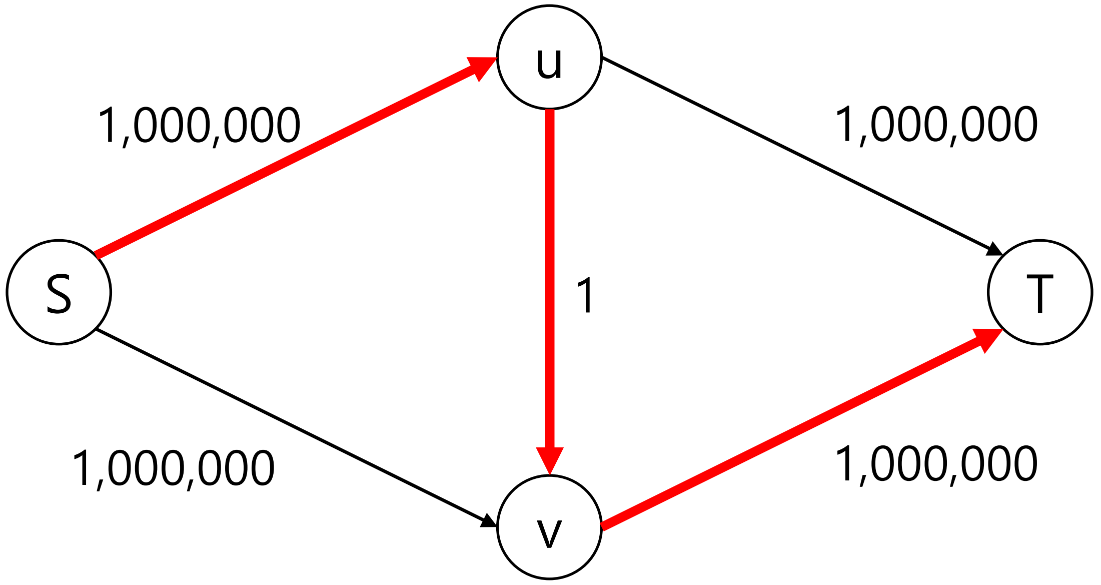
Ford-Fulkerson Algorithm

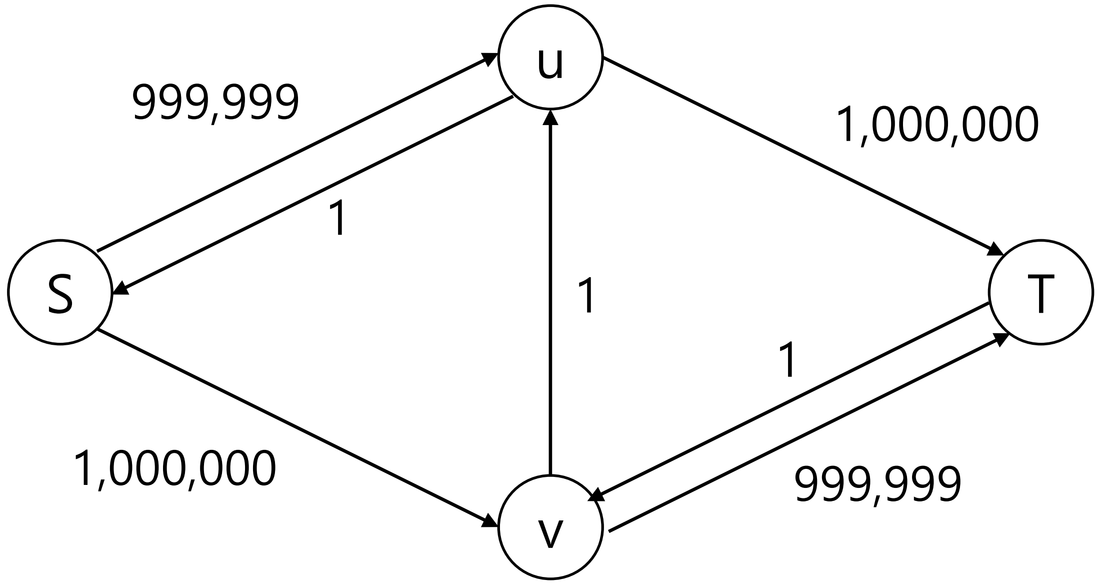
- Ford-Fulkerson Algorithm의 문제점



Ford-Fulkerson Algorithm

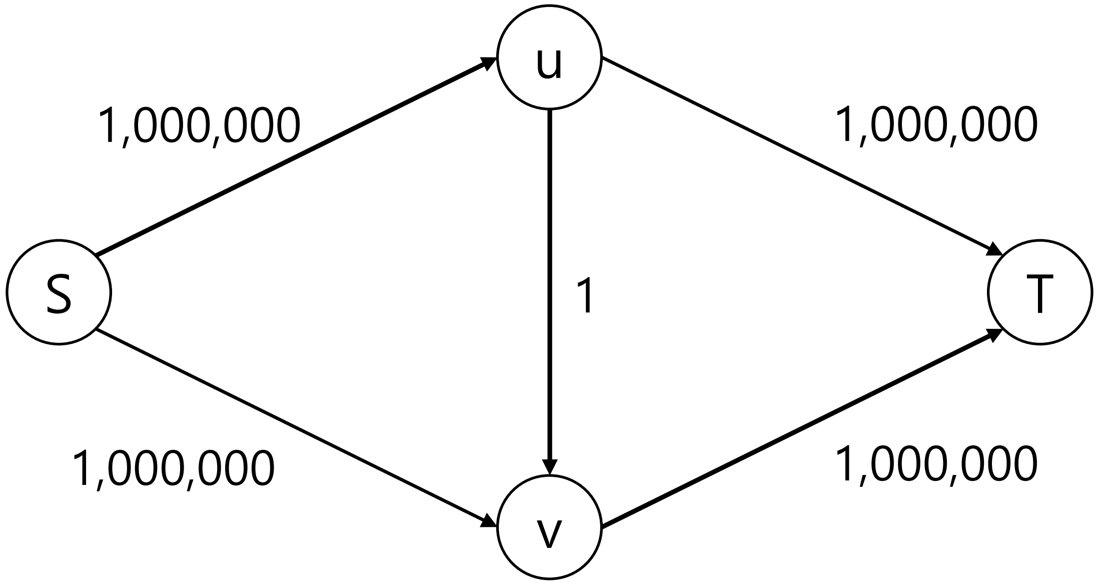
- Ford-Fulkerson Algorithm의 문제점

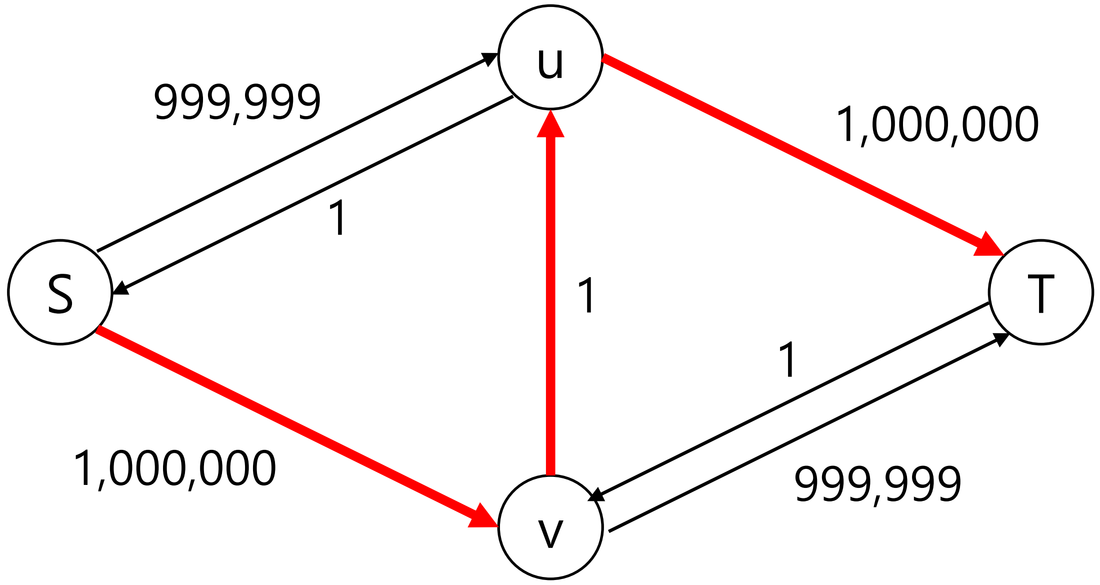




Ford-Fulkerson Algorithm

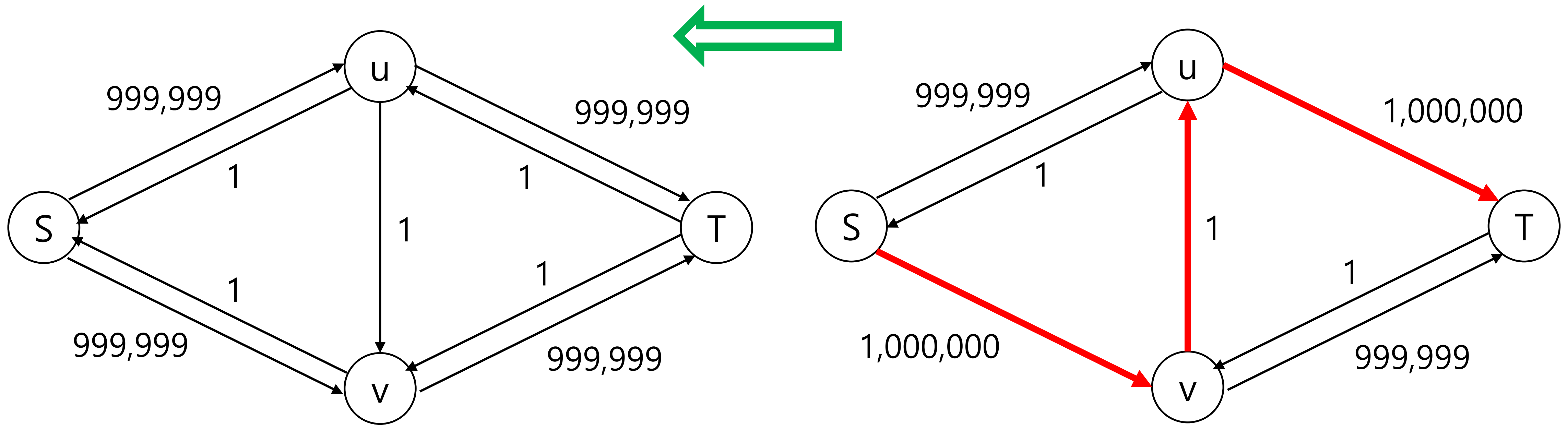
- Ford-Fulkerson Algorithm의 문제점





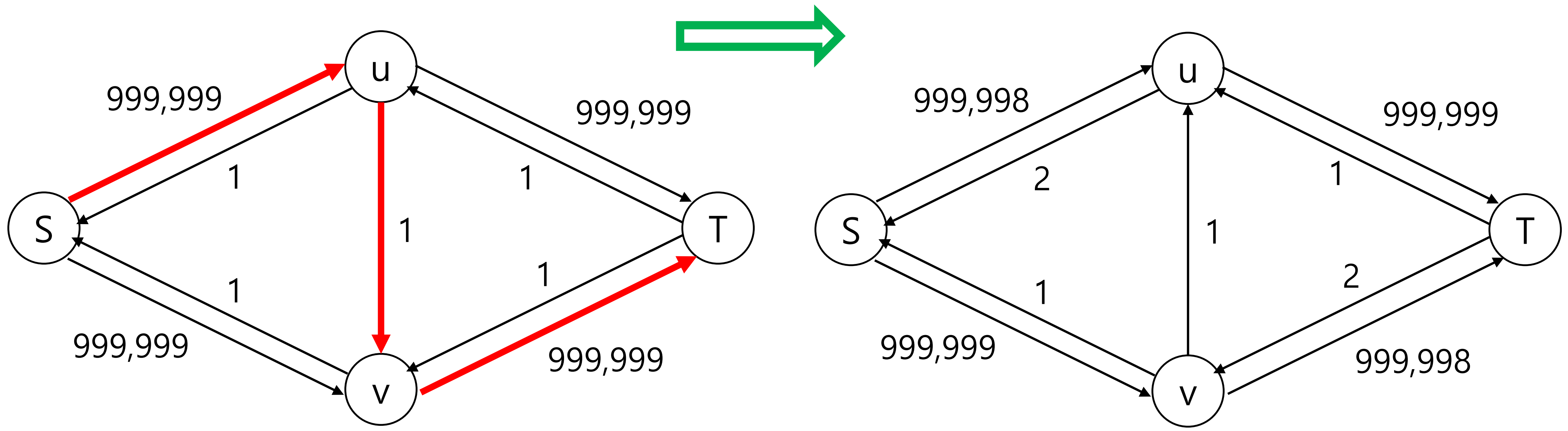
Ford-Fulkerson Algorithm

- Ford-Fulkerson Algorithm의 문제점



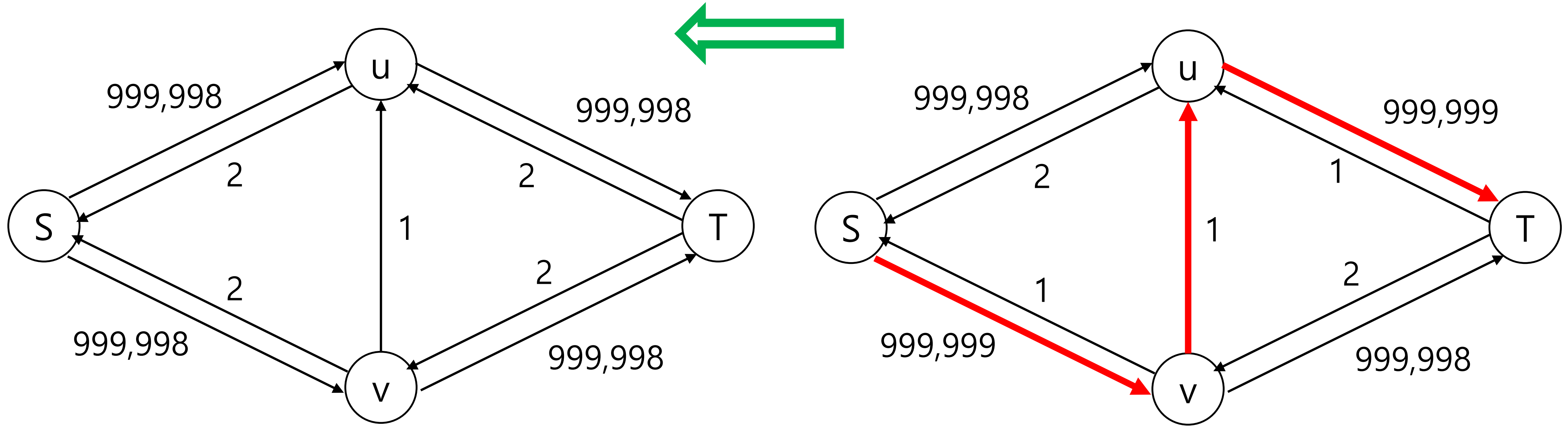
Ford-Fulkerson Algorithm

- Ford-Fulkerson Algorithm의 문제점



Ford-Fulkerson Algorithm

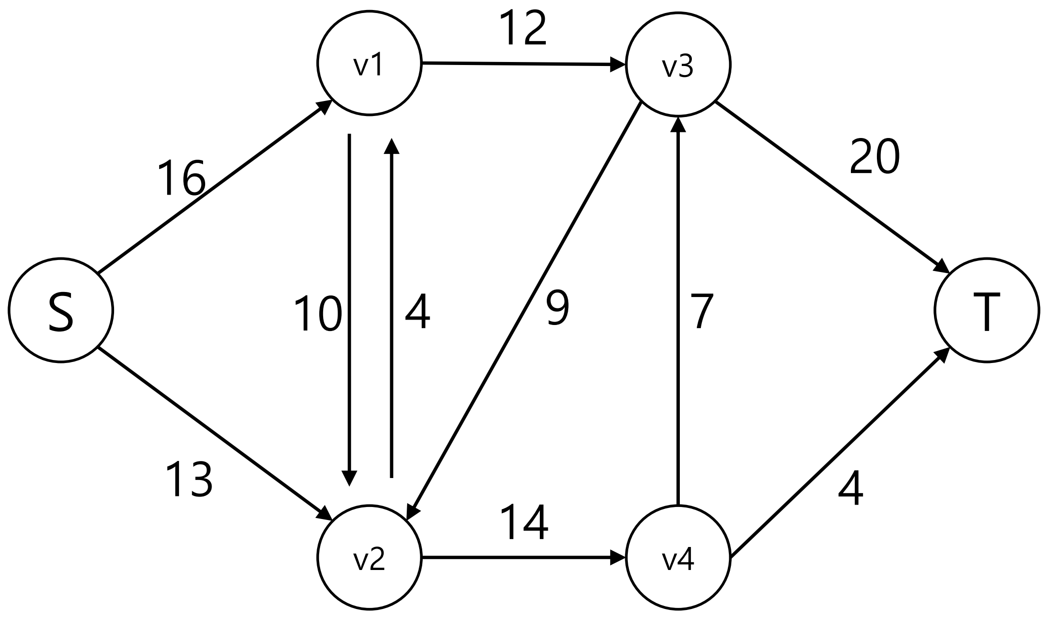
- Ford-Fulkerson Algorithm의 문제점



- $O(E|f^*|)$

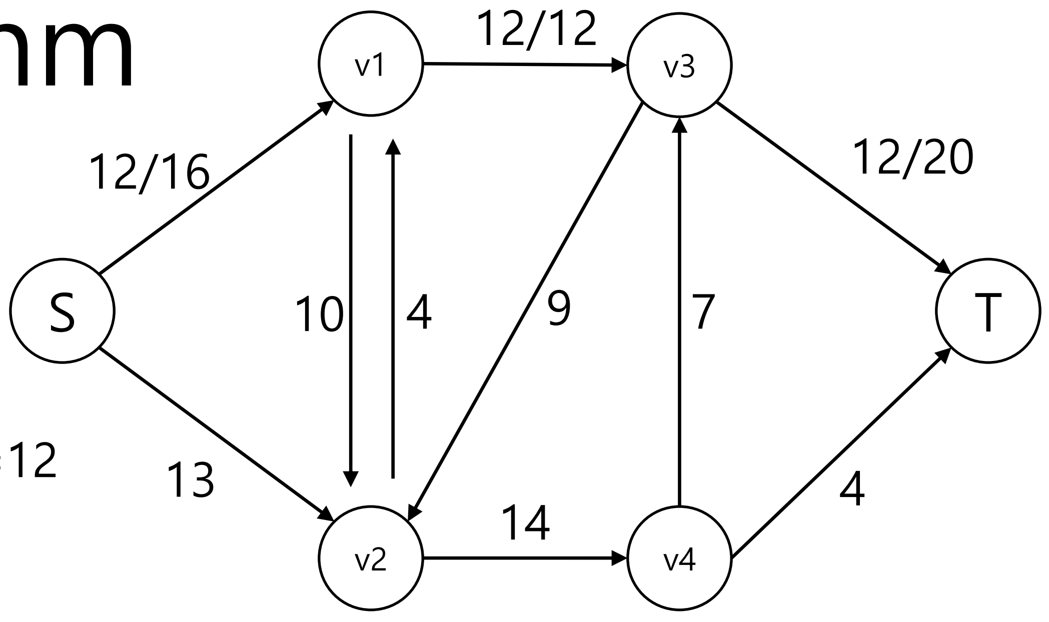
Edmonds-Karp Algorithm

- BFS로 augmenting path를 찾음

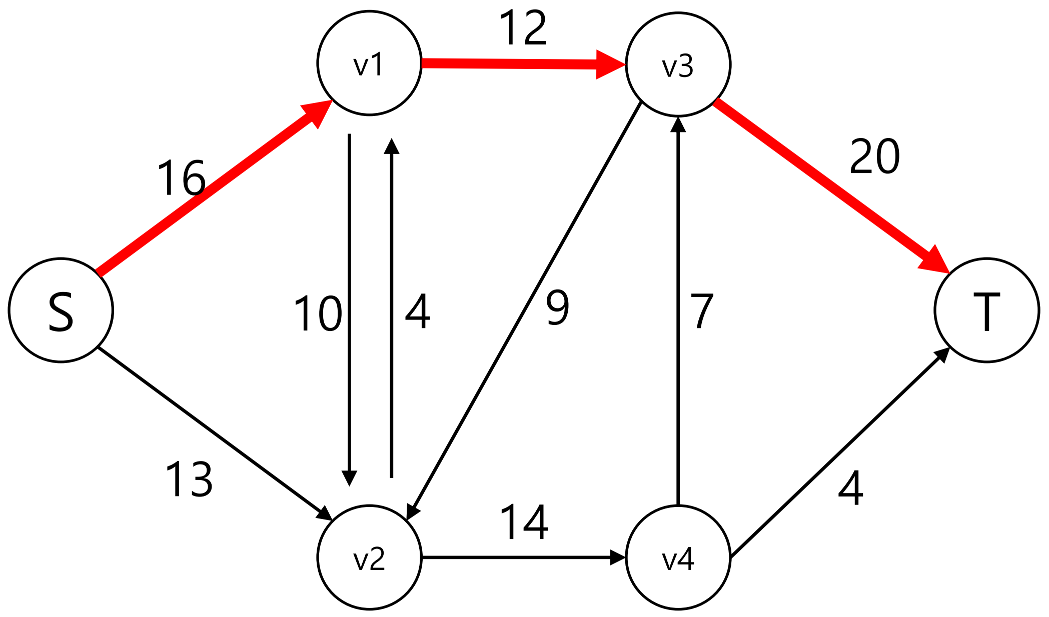


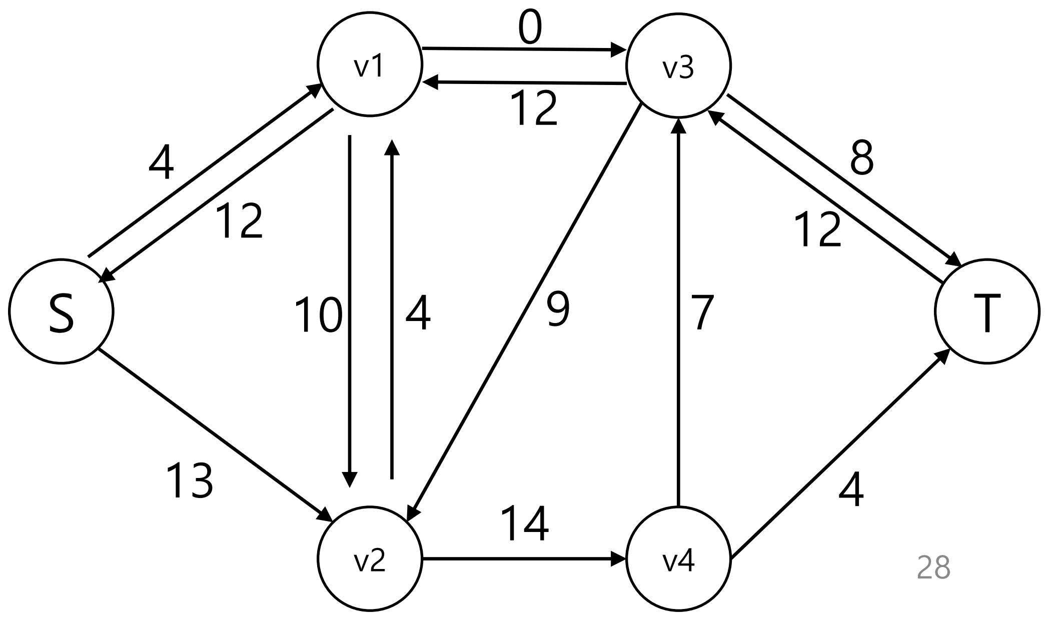
Edmonds-Karp Algorithm

- BFS로 augmenting path를 찾음



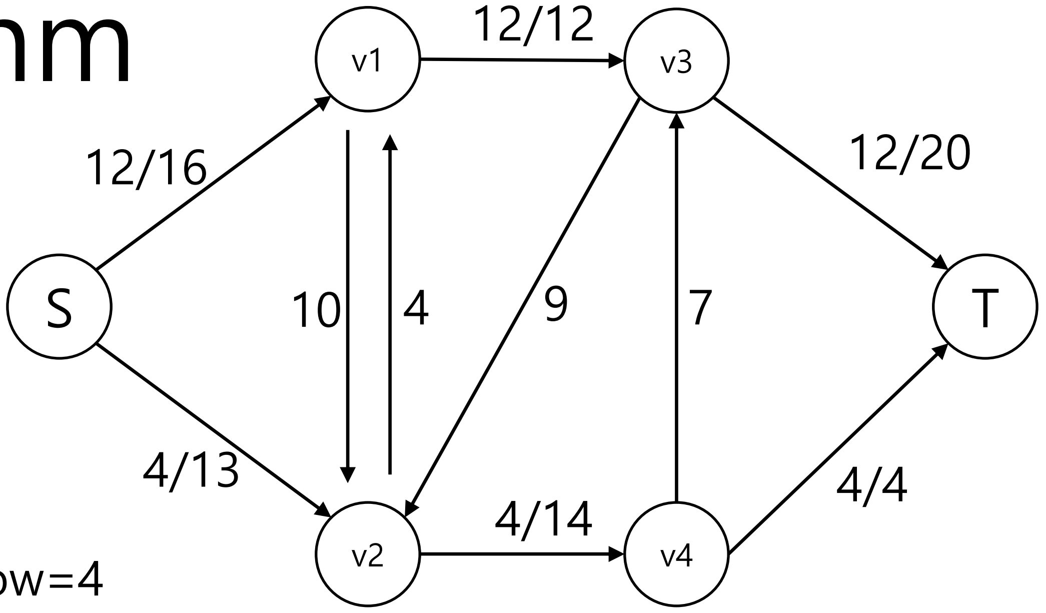
처음 찾은 augmenting path: flow=12



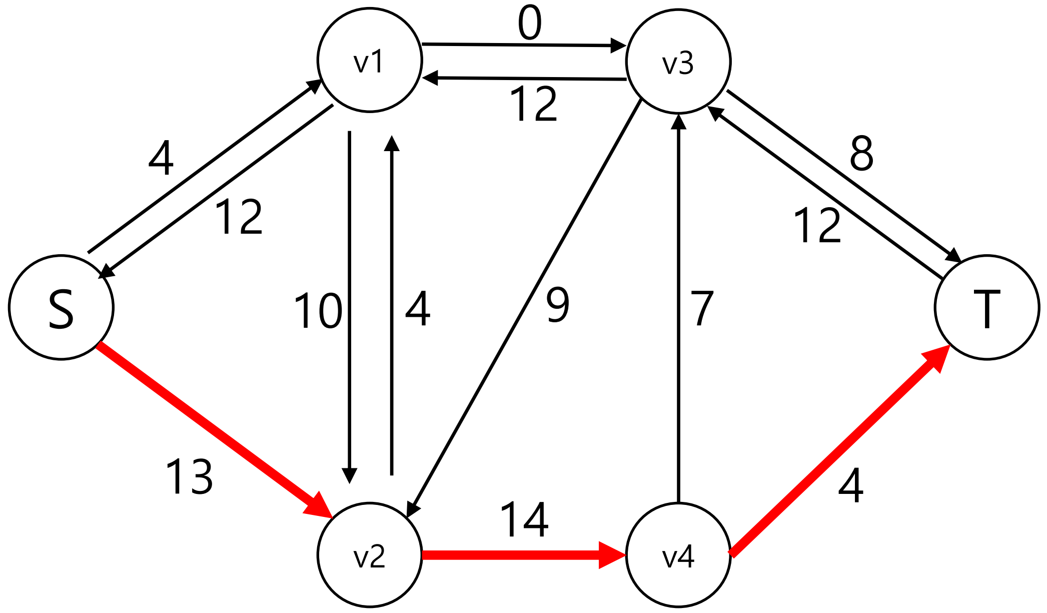


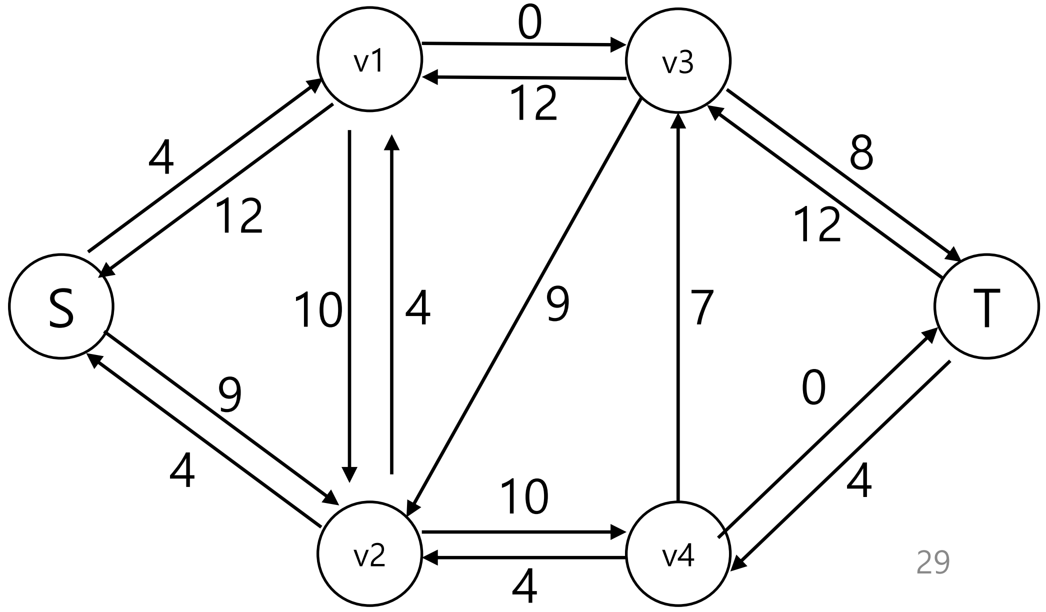
Edmonds-Karp Algorithm

- BFS로 augmenting path를 찾음



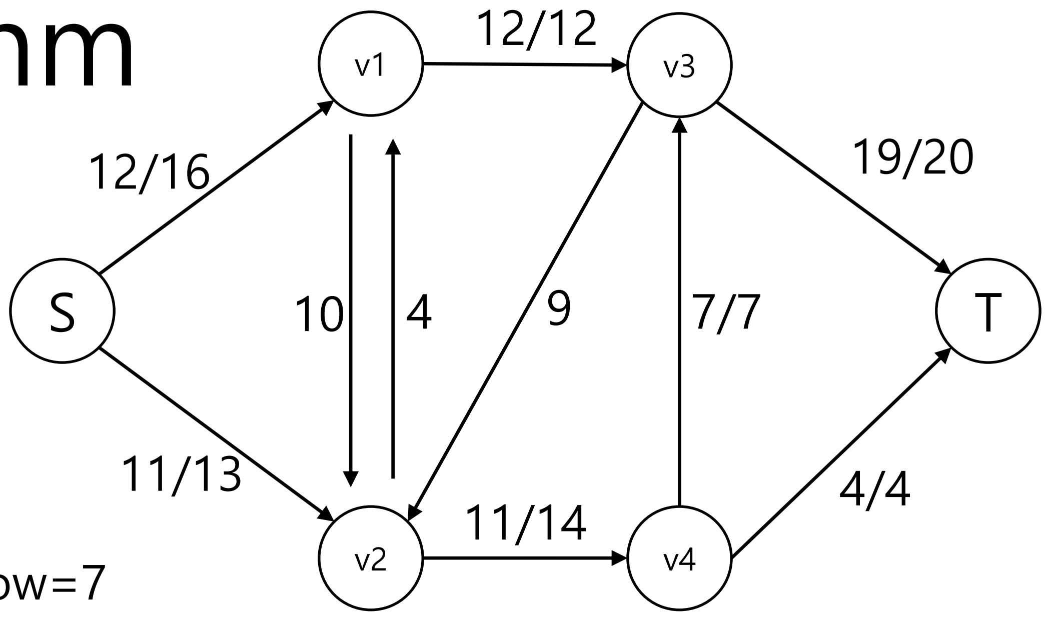
두번째 찾은 augmenting path: flow=4



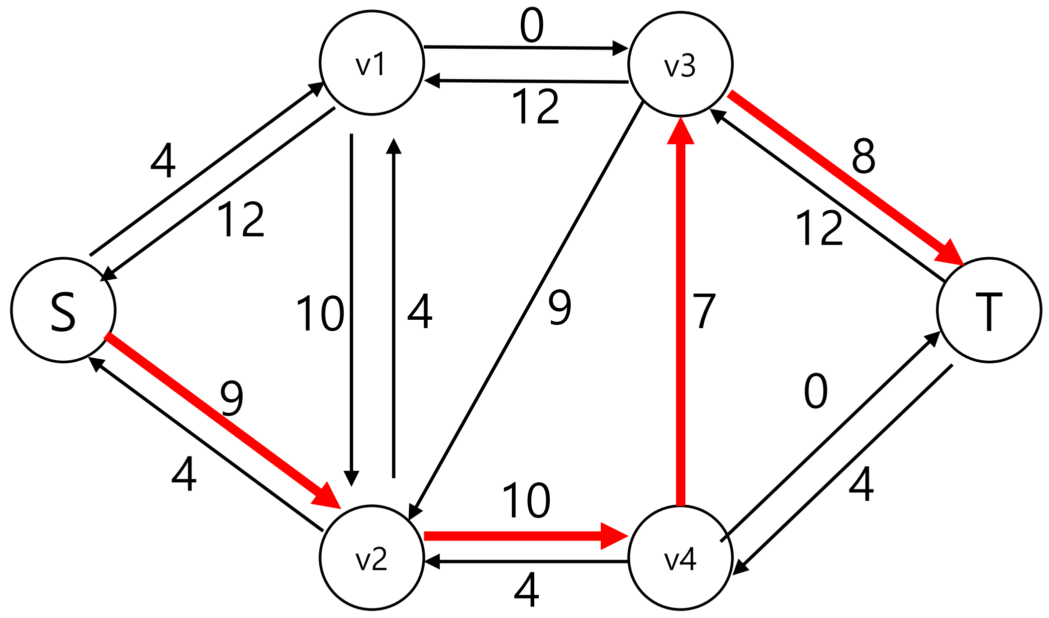


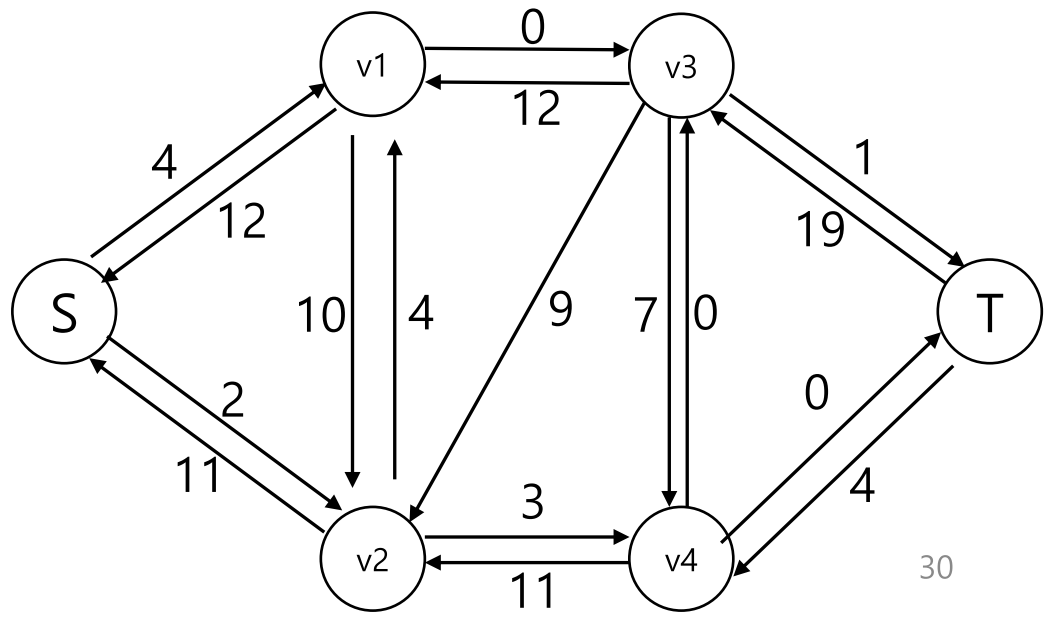
Edmonds-Karp Algorithm

- BFS로 augmenting path를 찾음



세번째 찾은 augmenting path: flow=7





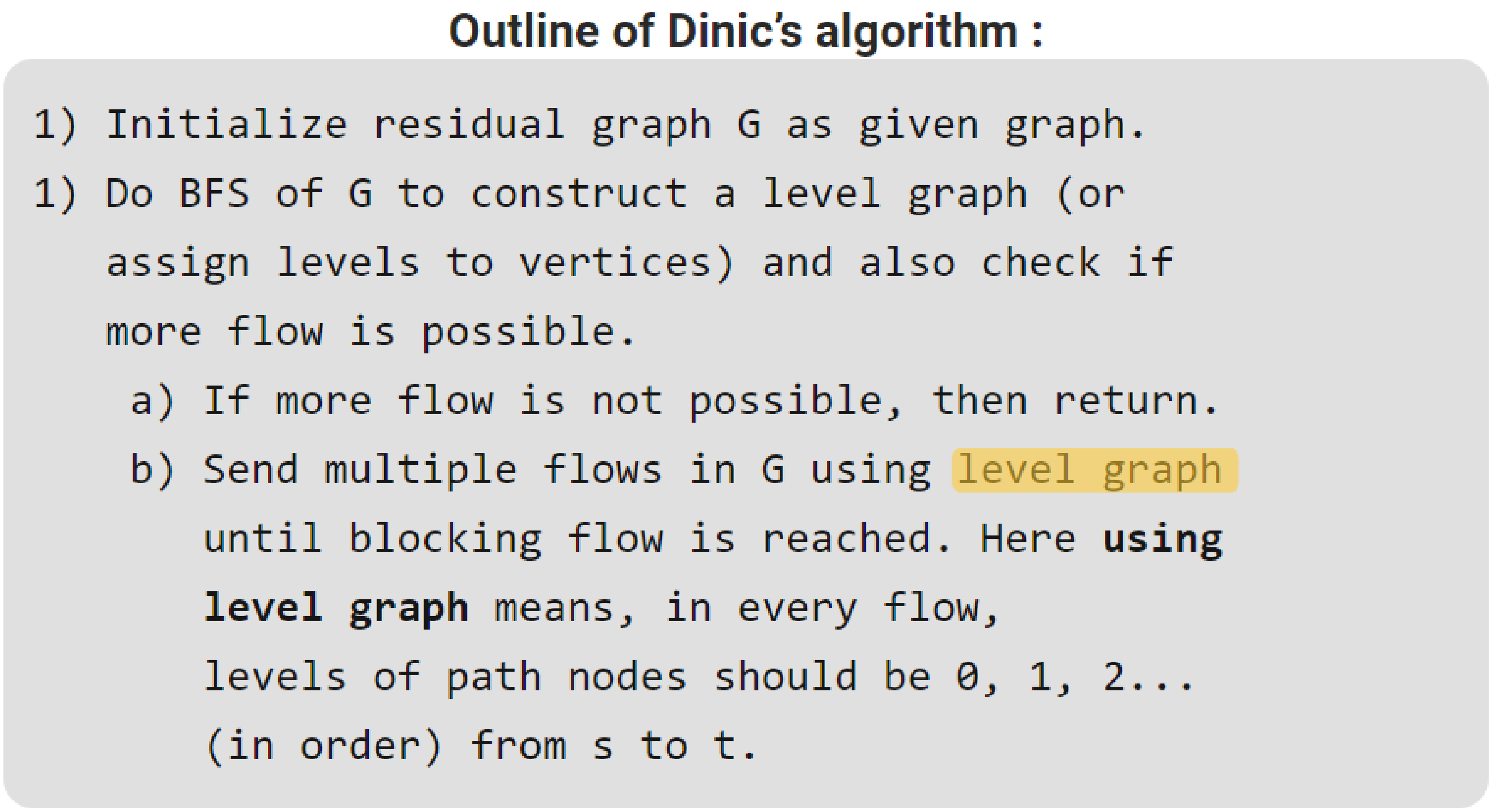
Edmonds-Karp Algorithm

- BFS로 augmenting path를 찾음
- BFS로 하나의 augmenting path를 찾는 시간: $O(E)$
- 모든 augmenting path를 찾기 위해 반복하는 횟수: $O(VE)$
- 따라서 이 알고리즘의 시간 복잡도: $O(VE^2)$

Dinic’s Algorithm

- https://www.geeksforgeeks.org/dinics-algorithm-maximum-flow/
- BFS 로 augmenting path를 찾음 (Edmonds-Karp 알고리즘 처럼)
- Edmonds-Karp 알고리즘과는 동작이 살짝 다름 ➔$O(EV^2)$

Dinic's Algorithm



Outline of Dinic's algorithm :

1) Initialize residual graph G as given graph.

1) Do BFS of G to construct a level graph (or assign levels to vertices) and also check if more flow is possible.

   a) If more flow is not possible, then return.

   b) Send multiple flows in G using level graph until blocking flow is reached. Here **using level graph** means, in every flow, levels of path nodes should be 0, 1, 2... (in order) from s to t.

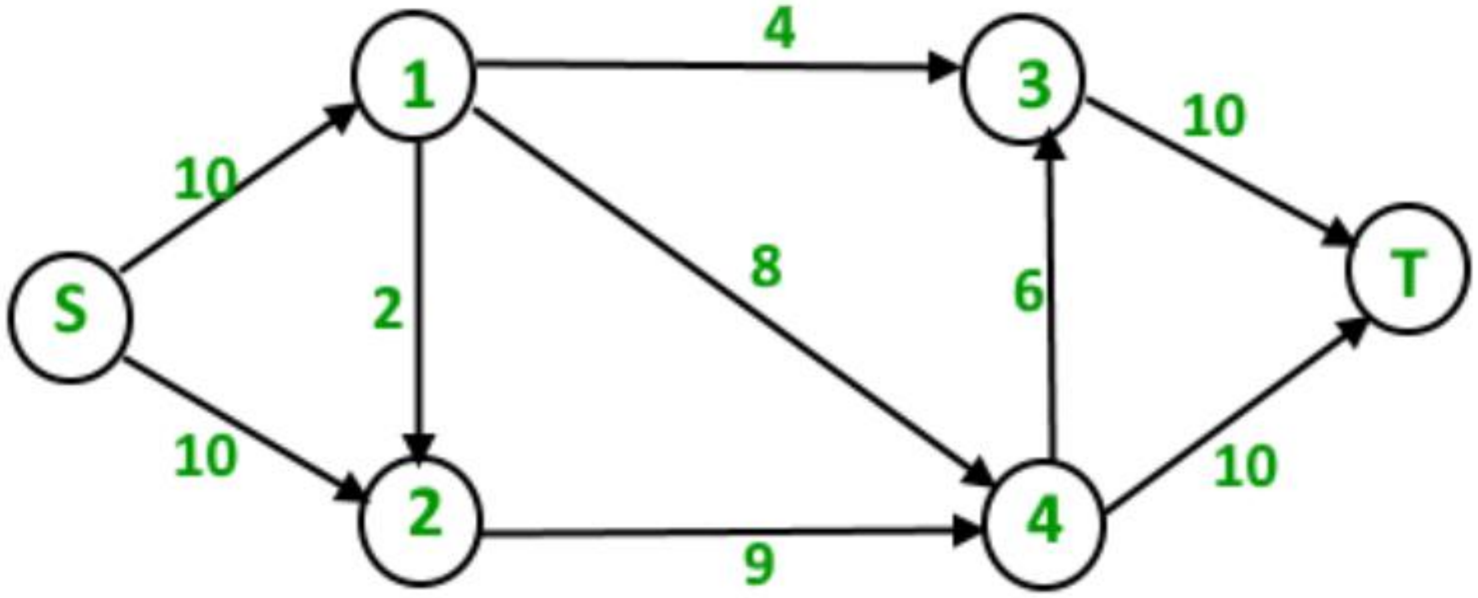
Dinic's Algorithm

- A flow is **Blocking Flow** if no more flow can be sent using level graph, i.e., no more s-t path exists such that path vertices have current levels 0, 1, 2… in order. Blocking Flow can be seen same as maximum flow path in Greedy algorithm

- 시간 복잡도: $O(EV^2)$

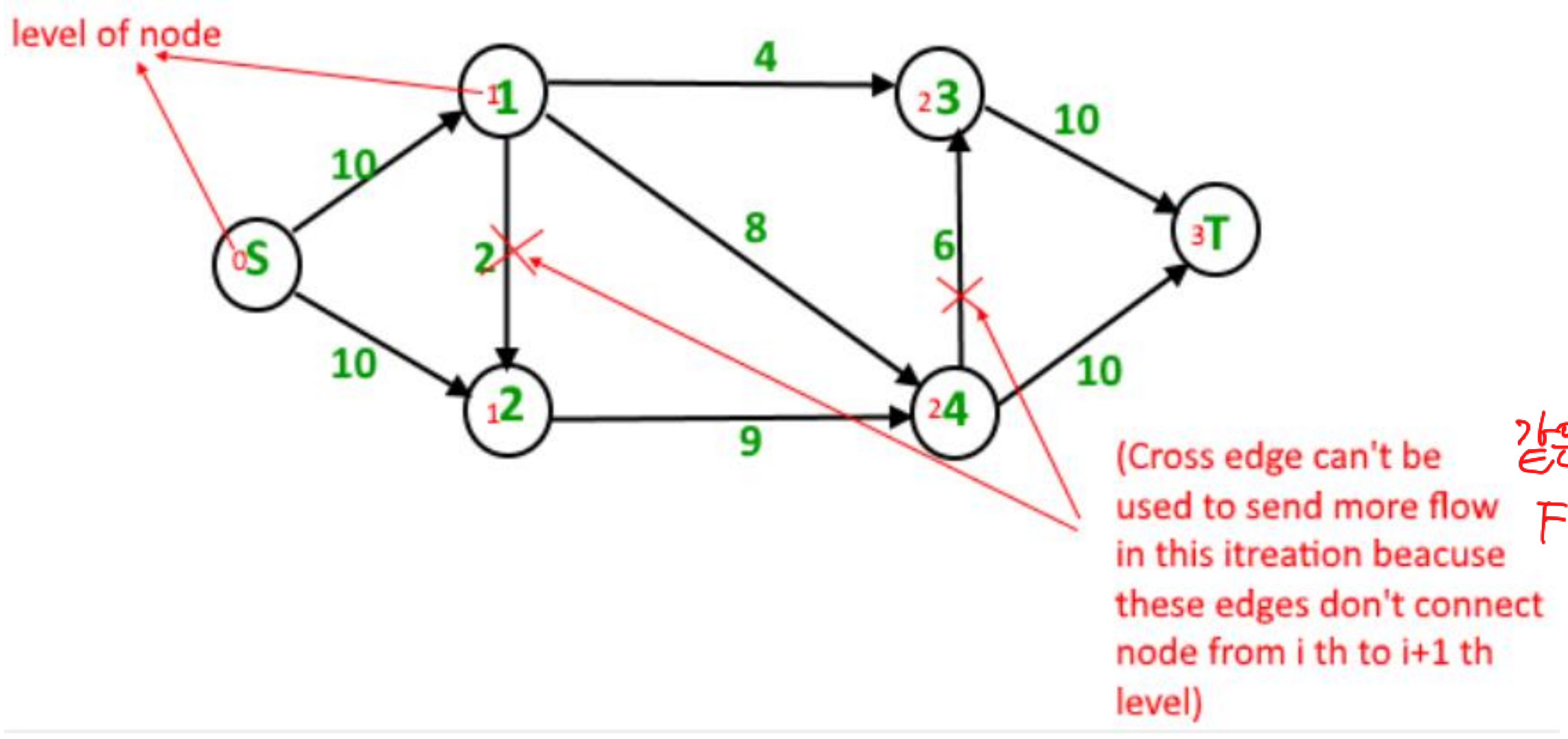
Dinic's Algorithm

- Initial residual graph



Dinic's Algorithm

- **First Iteration :** We assign levels to all nodes using BFS. We also check if more flow is possible (or there is a s-t path in residual graph).

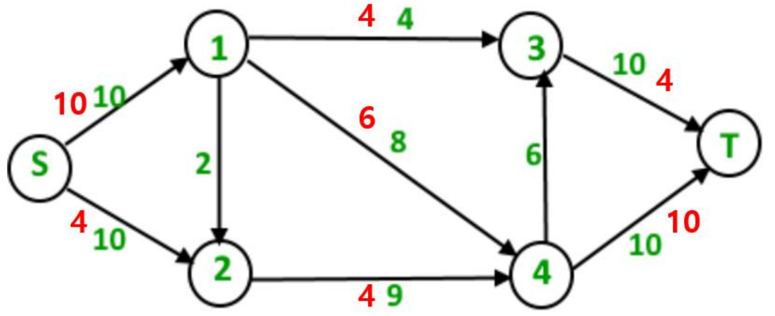


(Cross edge can't be used to send more flow in this itreation beacuse these edges don't connect node from i th to i+1 th level)

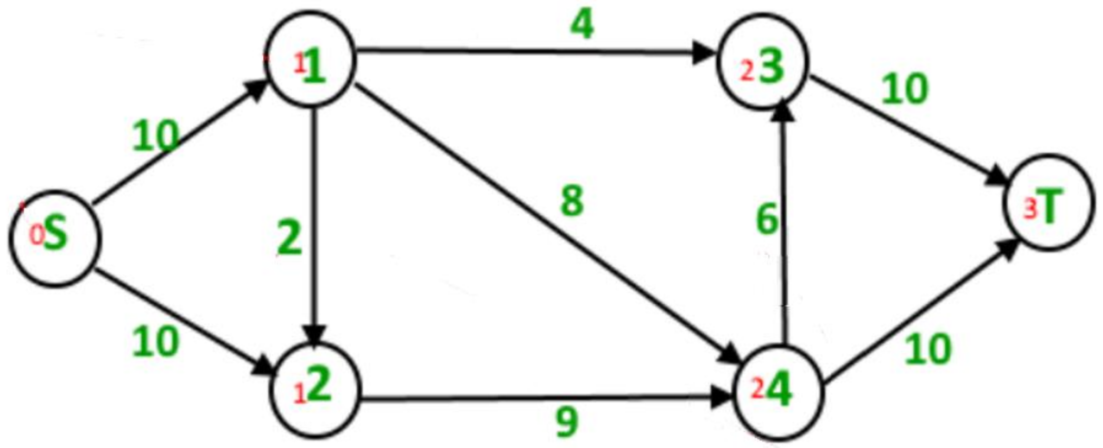
같은 레벨끼리 Flow 금지

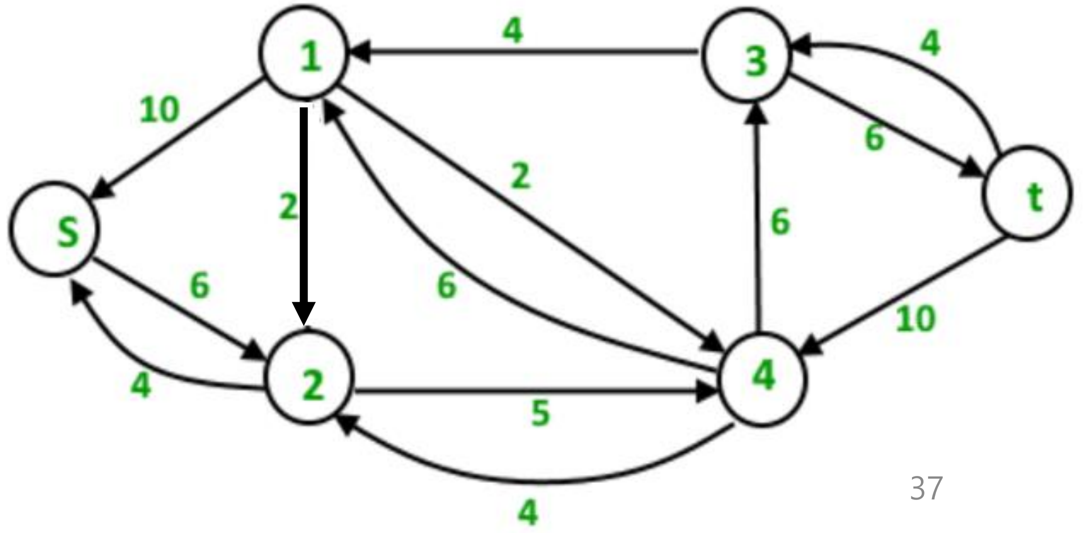
Dinic's Algorithm

- Now we find blocking flow using levels (means every flow path should have levels as 0, 1, 2, 3). We send three flows together. This is where it is optimized compared to Edmond Karp where we send one flow at a time.
  4 units of flow on path s – 1 – 3 – t. ①
  6 units of flow on path s – 1 – 4 – t. ②
  4 units of flow on path s – 2 – 4 – t. ③
  Total flow = Total flow + 4 + 6 + 4 = 14



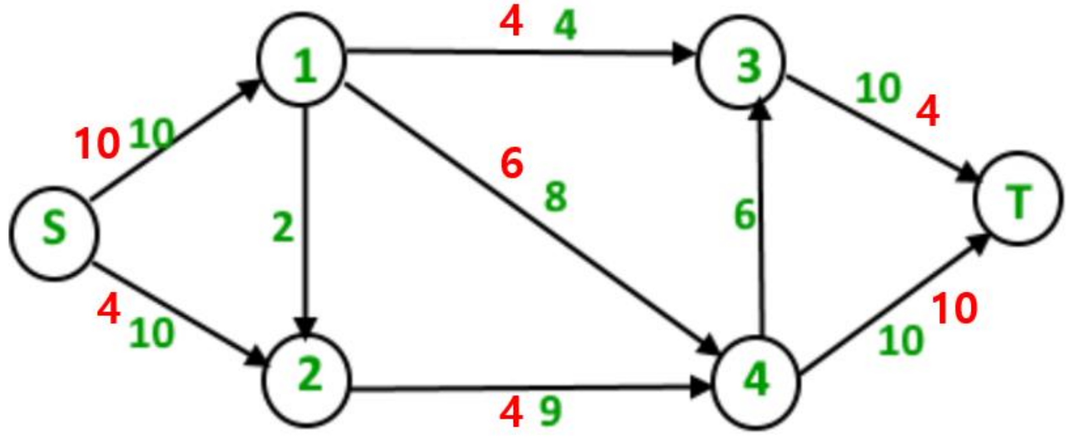
- After one iteration, residual graph changes to following.

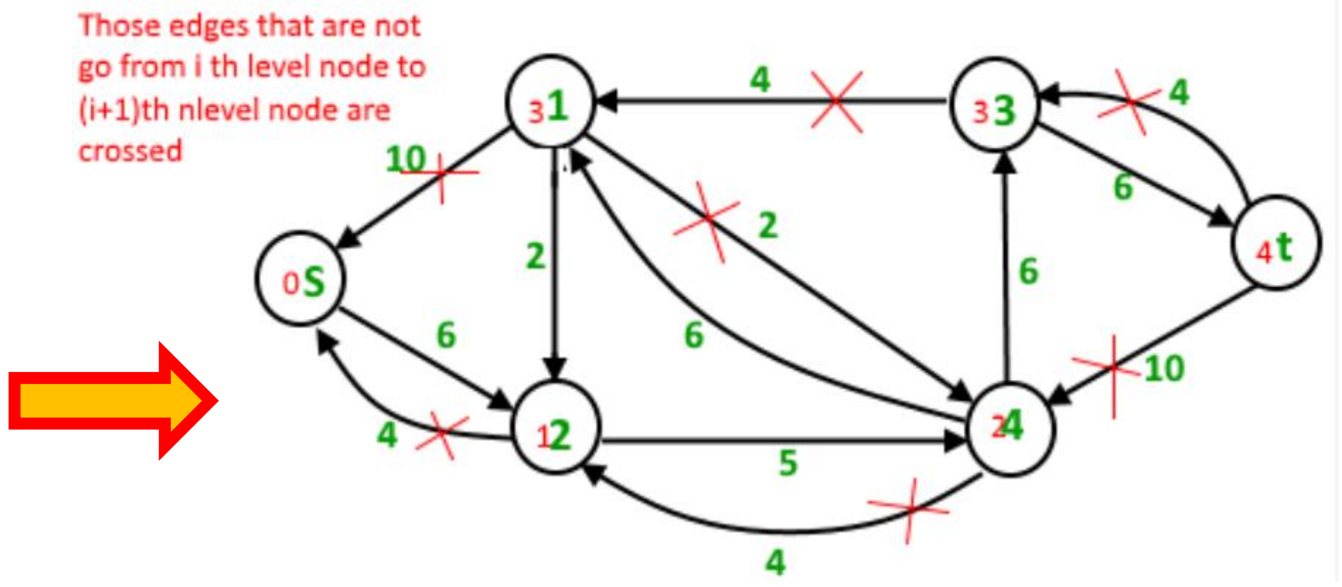




Dinic's Algorithm

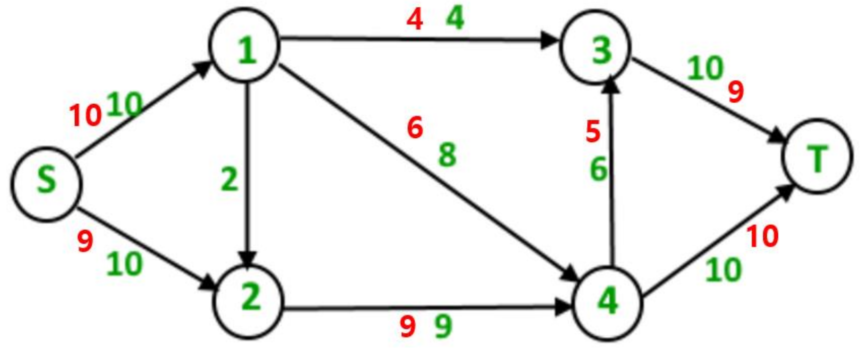
- **Second Iteration :** We assign new levels to all nodes using BFS of above modified residual graph. We also check if more flow is possible (or there is a s-t path in residual graph).

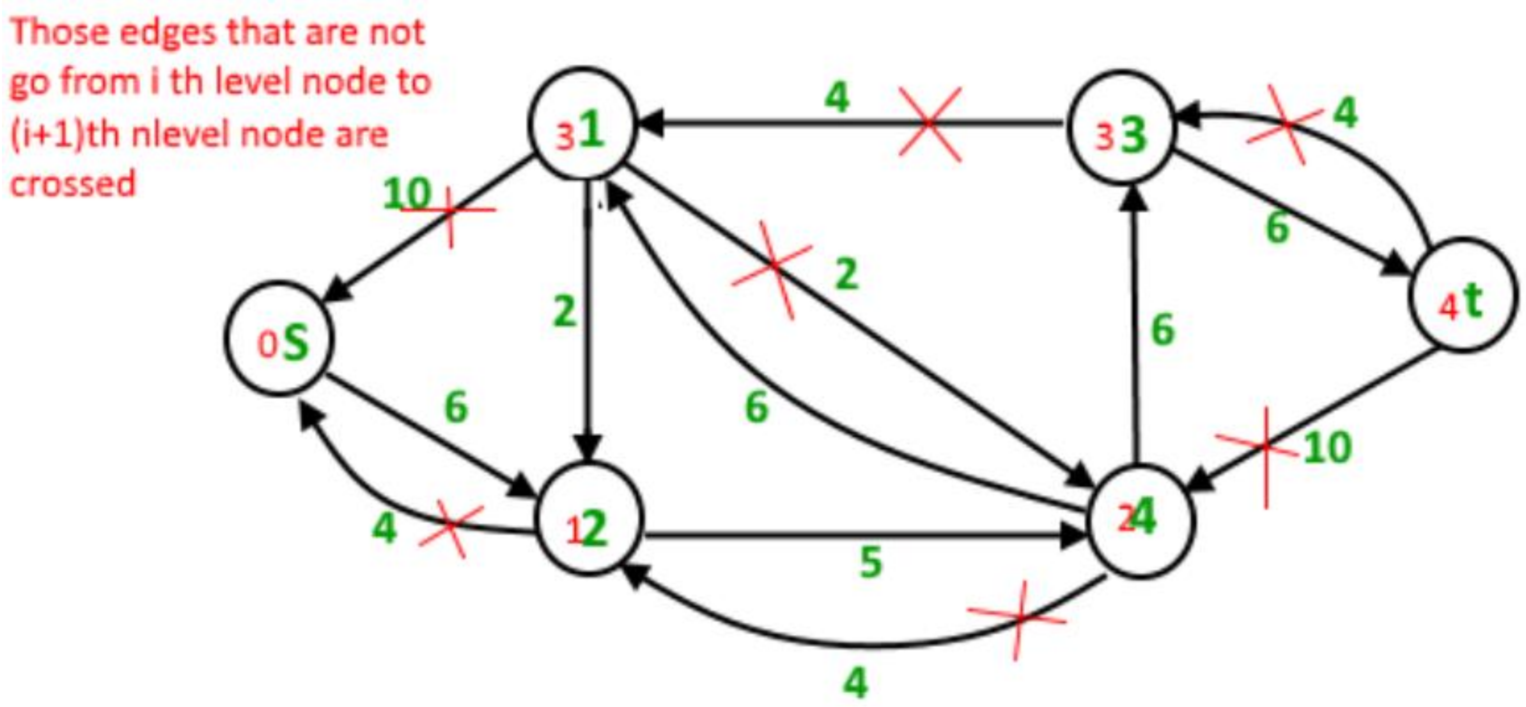



Those edges that are not go from i th level node to (i+1)th nlevel node are crossed
0 S
3 1
1 2
3 3
2 4
4 t
10
6
4
2
4
2
6
5
4
6
6
4
10

Dinic's Algorithm

- Now we find blocking flow using levels (means every flow path should have levels as 0, 1, 2, 3, 4). We can send only one flow this time.
  5 units of flow on path s – 2 – 4 – 3 – t
  Total flow = Total flow + 5 = 19

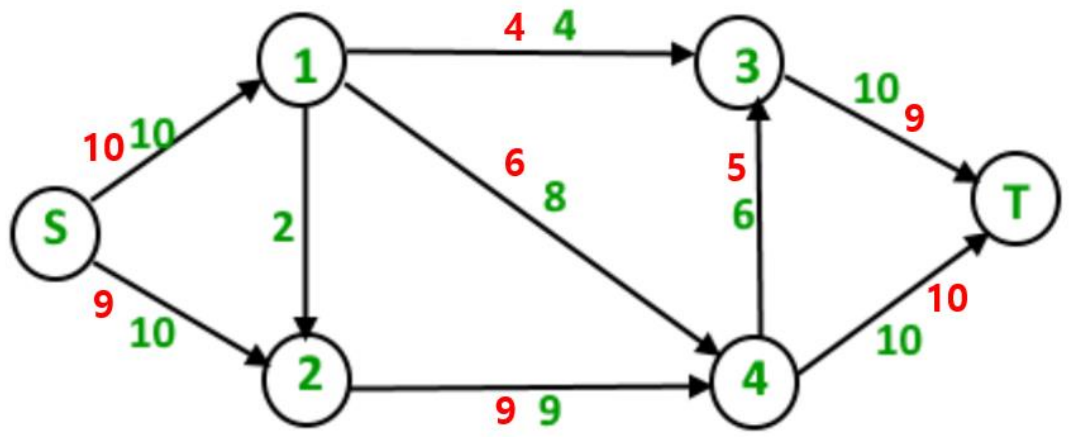


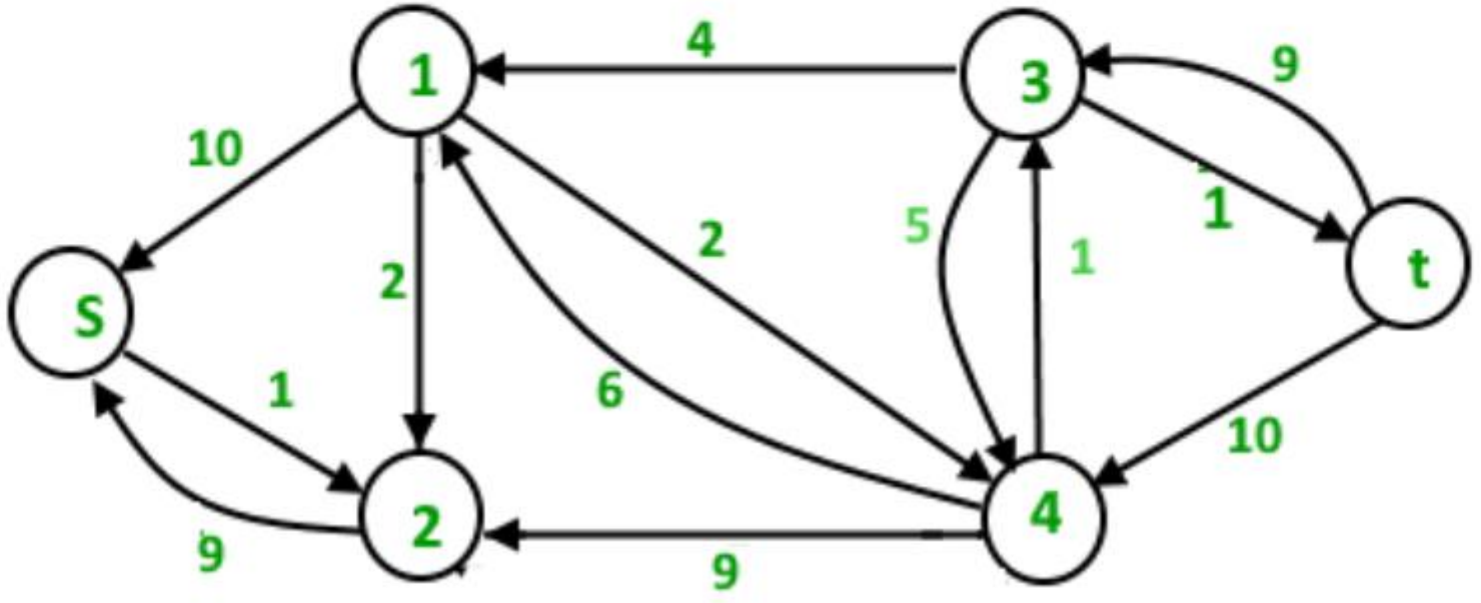


Those edges that are not go from i th level node to (i+1)th nlevel node are crossed
0S
1 2
3 1
2 4
3 3
4 t
10
2
6
4
6
2
4
5
4
6
6
4
10

Dinic's Algorithm

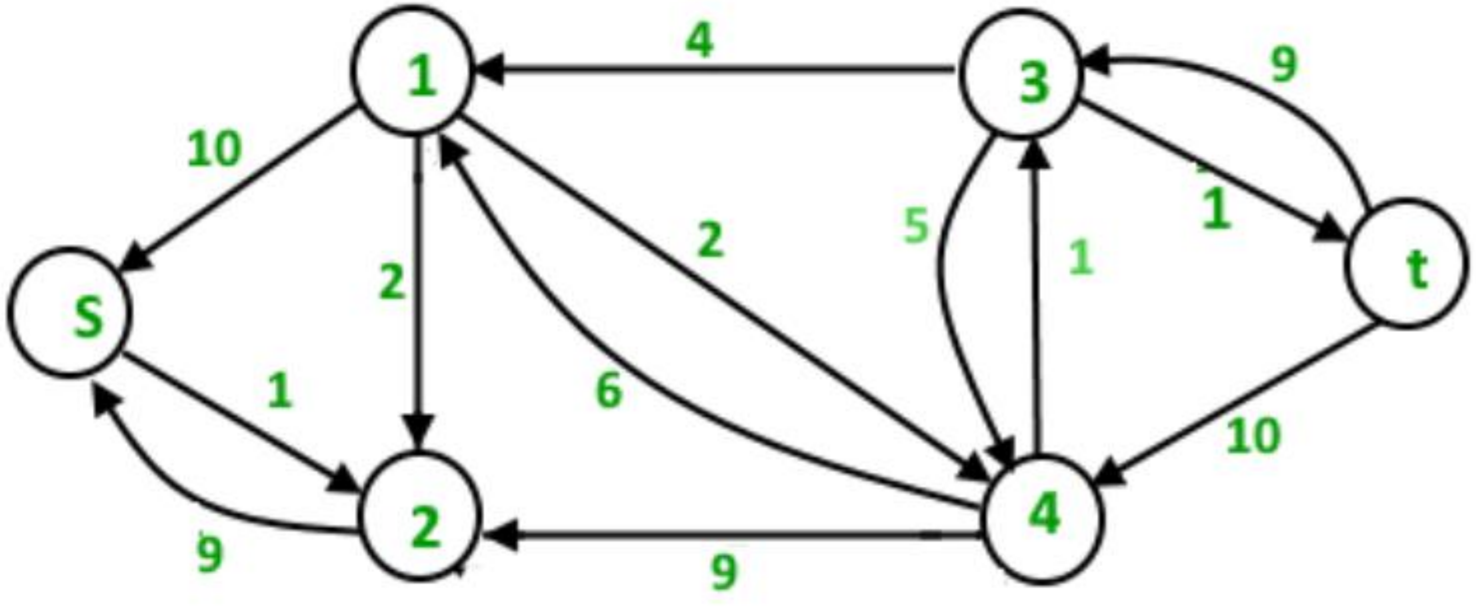
- The new residual graph is





Dinic's Algorithm

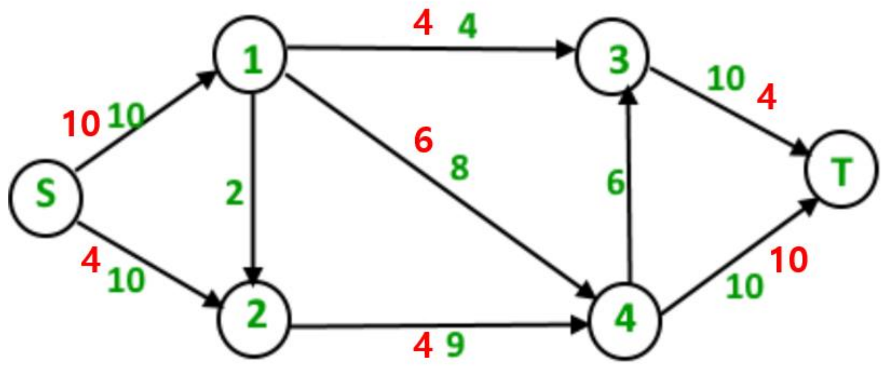
- **Third Iteration :** We run BFS and create a level graph. We also check if more flow is possible and proceed only if possible. This time there is no s-t path in residual graph, so we terminate the algorithm.



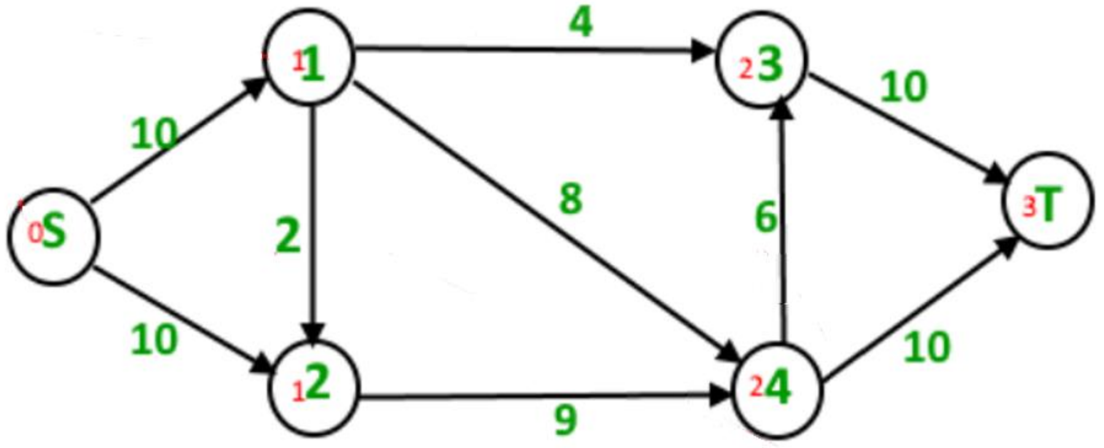
Comments on Dinic's Algorithm

- BFS 처리 후, 만약 앞에서 찾은 경로 순서와 다르게 처리하면?
  (5장 이전 슬라이드와 비교)
- 8 units of flow on path s – 1 – 4 – t.
  2 units of flow on path s – 1 – 3 – t.
  2 units of flow on path s – 2 – 4 – t.
  Total flow = 12

- 이후의 처리는 각자 해 본 후 비교해 보자.

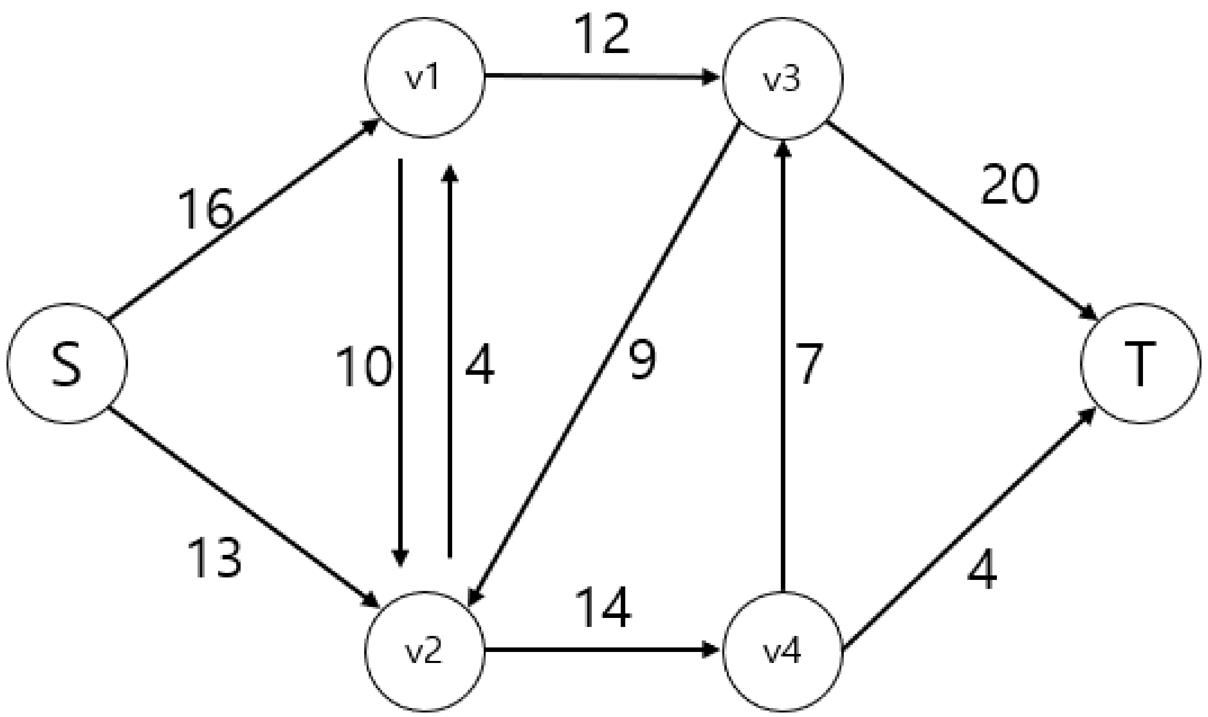


5장 이전 슬라이드에 보인 결과



Max-flow min-cut theorem

- What is a cut ?
- How to find a min-cut ?



S→T 의 flow를 끊어내는

최소비용

Max-flow min-cut theorem

- If f is a flow network $G = (V, E)$ with source s and sink t, then the following conditions are equivalent:
  1. f is a maximum flow in G.
  2. The residual network G_f contains no augmenting paths.
  3. $|f| = c(S, T)$ for some cut (S, T) of G

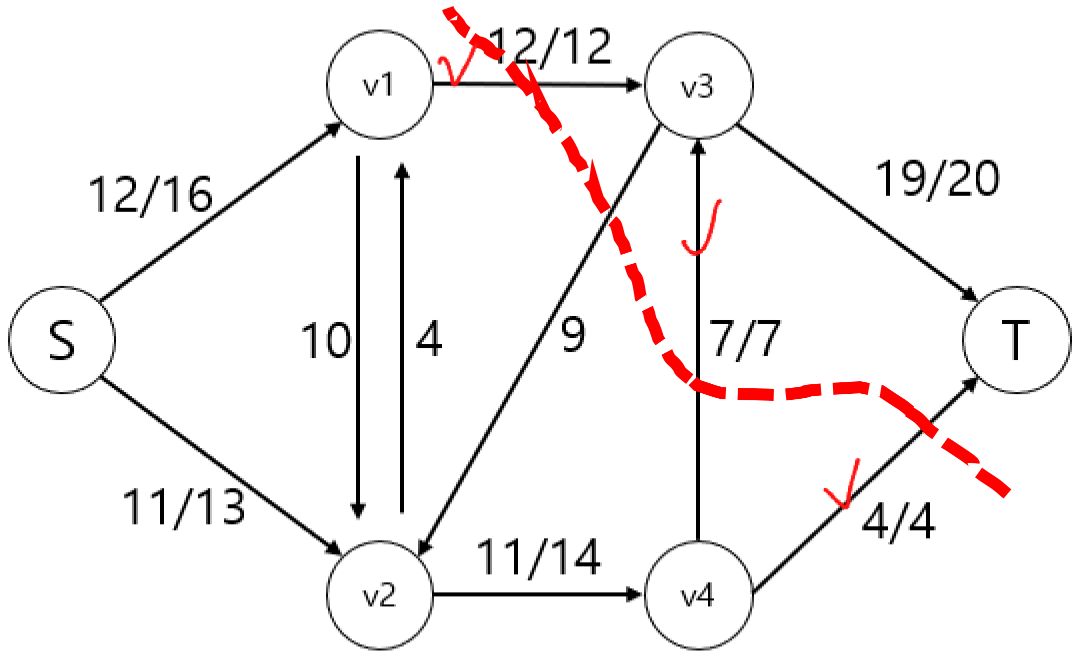
Max-flow min-cut theorem

- If f is a flow network $G = (V, E)$ with source s and sink t, then the following conditions are equivalent:
  1. f is a maximum flow in G.
  2. The residual network G_f contains no augmenting paths.
  3. $|f| = c(S, T)$ for some cut (S, T) of G

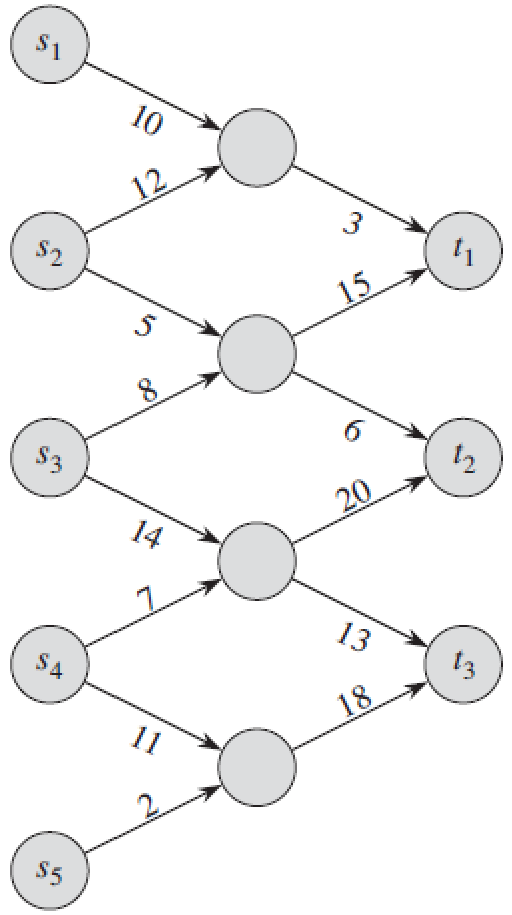
- How to find a min-cut

- What if given graph is undirected?

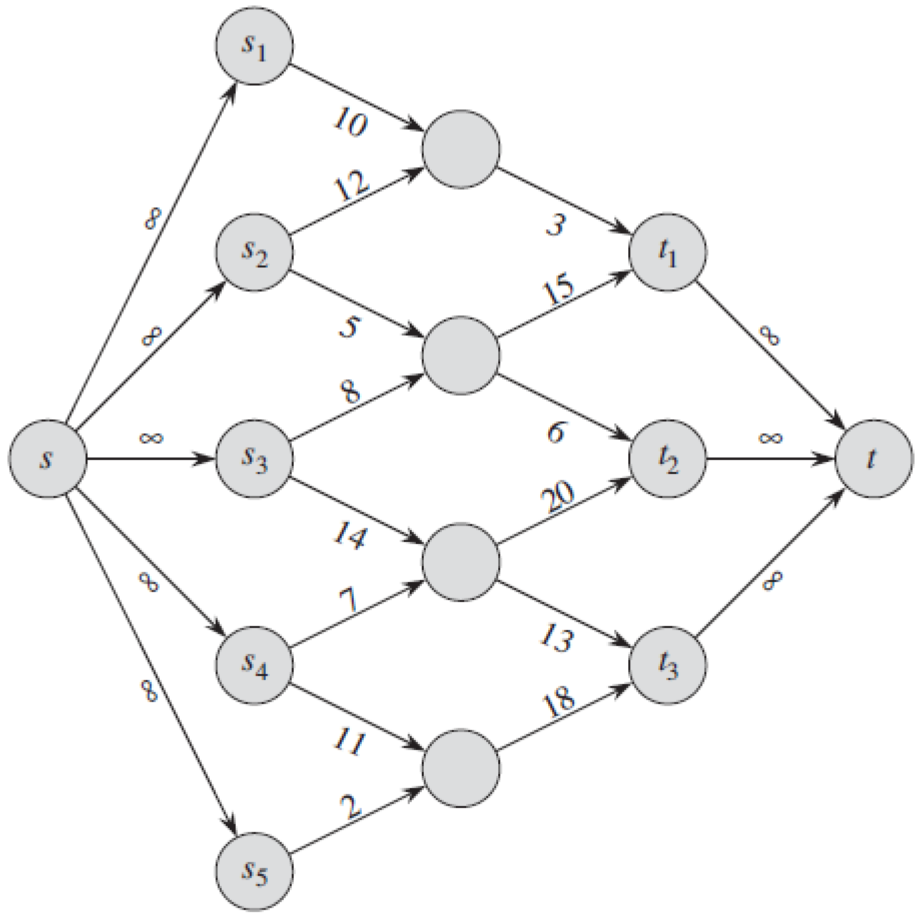
$\Rightarrow$ 양방향으로 설정



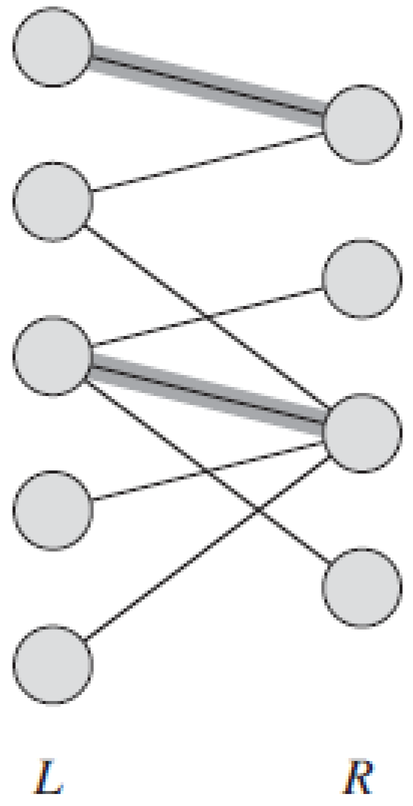
Networks with multiple sources and sinks

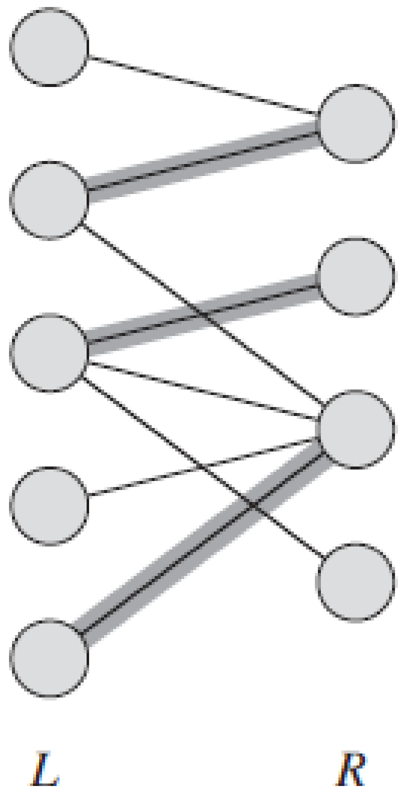


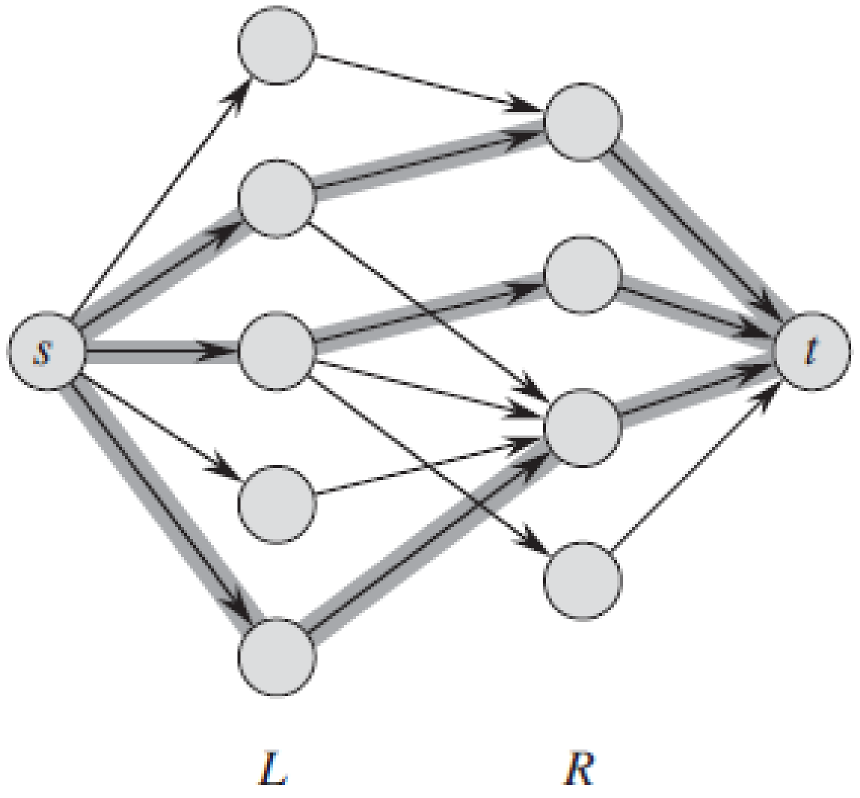
$\Rightarrow$



Maximum bipartite matching







Problem E
Jerry and Tom
Time Limit: 1 Second

Naughty mouse Jerry and his friend mice sometimes visit a vacant house to play the famous children game 'hide and seek' and also to adjust the length of their teeth by gnawing furniture and chairs left there. If we look down the house from the sky the boundary of it composes an orthogonal polygon parallel to the xy-axes as shown in the figure below. In other words, every wall of the house is either horizontal or vertical.

Tom, a threatening cat to them, sometimes appears in the house while they are enjoying the game. In that case Jerry and his friends should hide into the rat's holes at the bottom on the walls. There are two rules which must be held for them to hide into the holes:

1. Each hole can afford at most k mice.
2. Each mouse can enter the hole which can be seen by it. In other words, a mouse cannot enter the hole which is hidden by any wall. (That is, if the connecting line between a mouse and a hole intersects either any wall or any corner point of the house, the hole is considered hidden from the mouse.)

For example, consider a situation where three mice and three holes are in the house as shown in Figure E.1. Each circle on the boundary denotes a hole. Assuming that $k = 1$, i.e., only one mouse is allowed to hide into each hole, with the situation shown in the left figure, when Tom appears all the three mice can hide. But for the case shown in the right figure it is impossible for all the mice to hide.

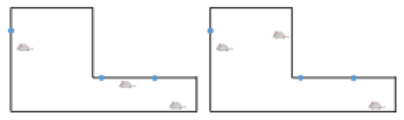


Figure E.1: Illustration to show two situations: 1. All mice can hide (left) and 2. They cannot (right)

You can assume:

1. Every mouse is strictly inside the house, which means that no mouse is on the wall
2. Every hole is on the wall.
3. No two holes locate at the same spot.
4. No two mice locate at the same spot.

Given a situation explained above, you are to write a program which determines whether all the mice can hide or not.

Input
Your program is to read from standard input. The input starts with a line containing four integers, n, k, h, and m, where $n(1 \le n \le 1{,}000)$ is the number of the corner points of a house, $k(1 \le k \le 5)$ the maximum number of mice each hole can afford, $h(1 \le h \le 50)$ the number of holes, $m(1 \le m \le k \cdot h)$ the number of mice. In each of the following n lines, each coordinate of the corner points of the house is given in counter clockwise order. Each point is represented by two integers separated by a single space, which are the x-coordinate and the y-coordinate of the point, respectively. Each coordinate is given as an integer between -10^9 and 10^9, inclusively. In each of the following h lines, two integers x and y are given, which represent the coordinate (x, y) of each hole. In each of the following m lines, two integers x and y are given, which represent the coordinate (x, y) of each mouse.

Output
Your program is to write to standard output. Print exactly one line for the input. Print `Possible` if all the mice can hide into the rat's holes holding the constraints explained above. Otherwise print `Impossible`.

The following shows sample input and output for two test cases.

Sample Input 1	Output for the Sample Input 1

6 1 3 3
0 0
100 0
100 50
40 50
40 70
0 70
0 55
55 50
80 50
15 65
90 10
92 10
	Possible

Sample Input 2	Output for the Sample Input 2

6 1 3 3
0 0
100 0
100 50
40 50
40 70
0 70
0 55
55 50
80 50
15 65
90 10
30 66
	Impossible

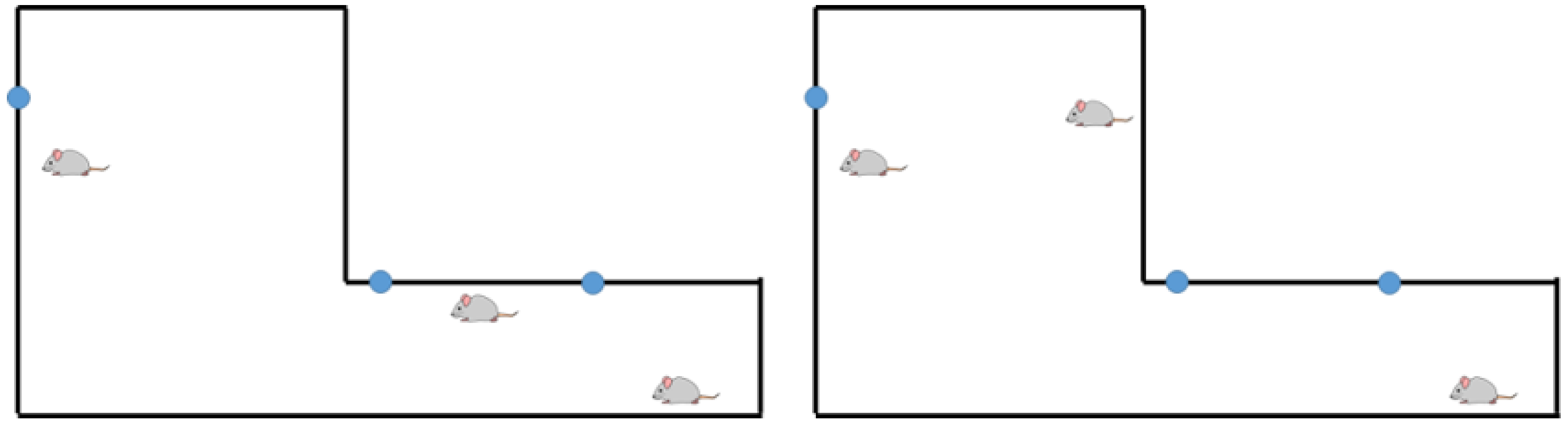
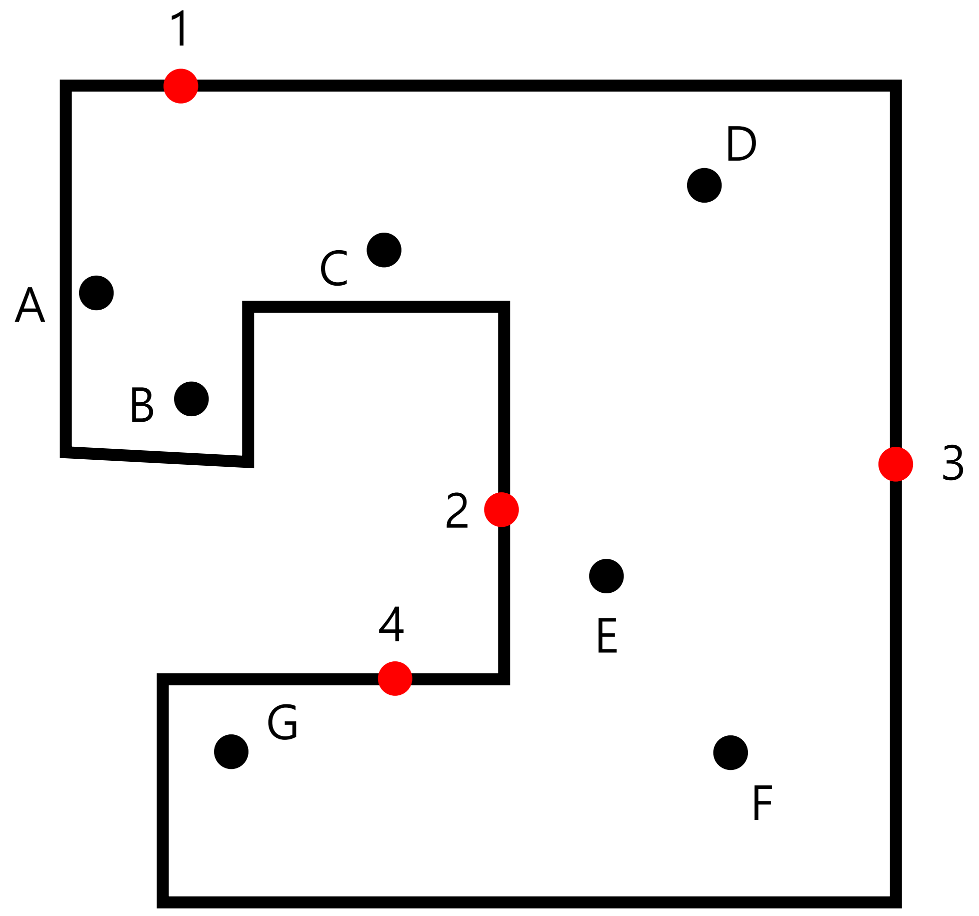


Figure E.1: Illustration to show two situations: 1. All mice can hide (left) and 2. They cannot (right)

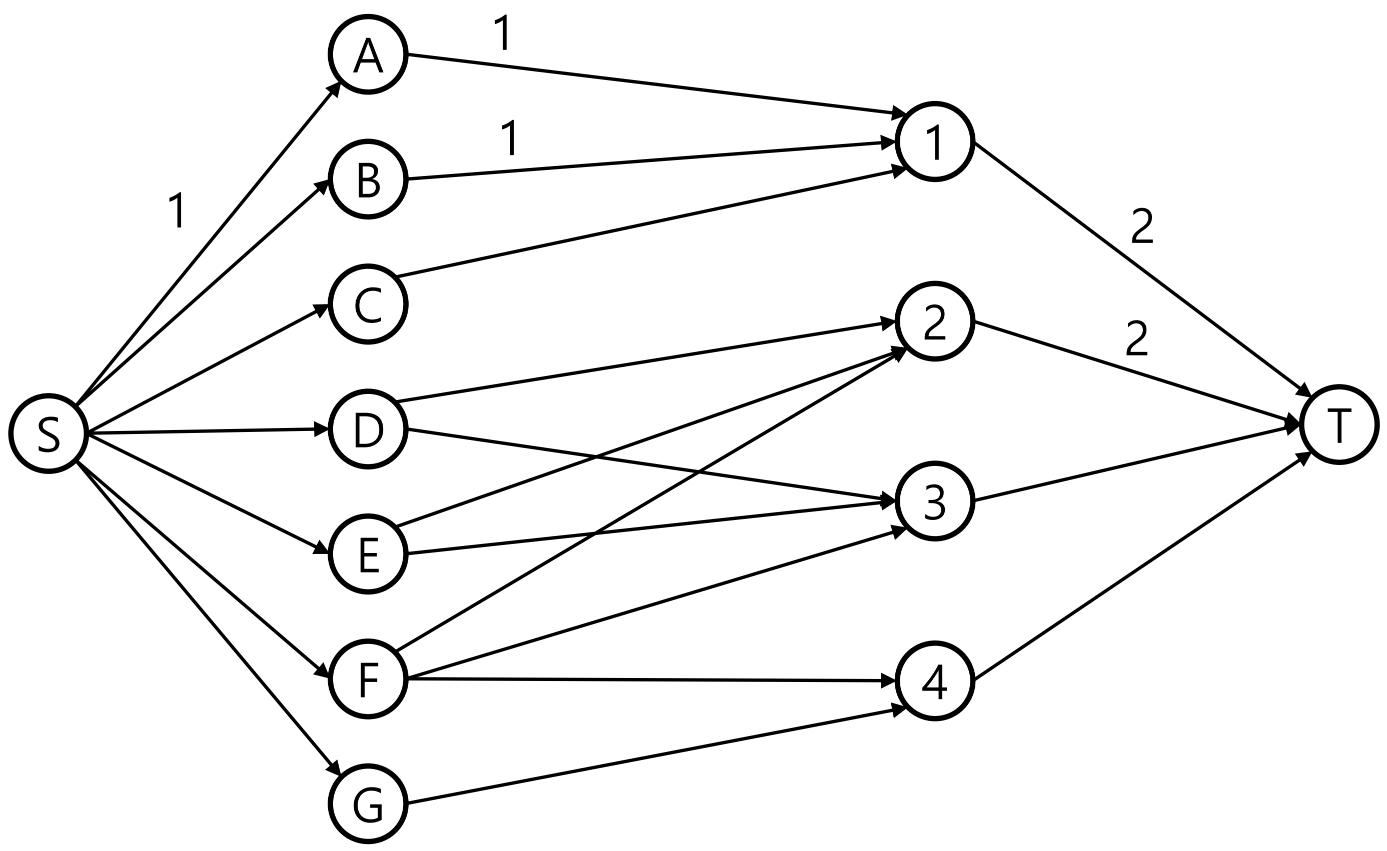
문제를 풀기 위한 모델링

- 이런 류의 문제를 풀기 위해선 문제에 적합하게 그래프를 설계해야 한다.
- 어떻게? (5분간 각자 생각해 봅시다)

문제를 풀기 위한 모델링



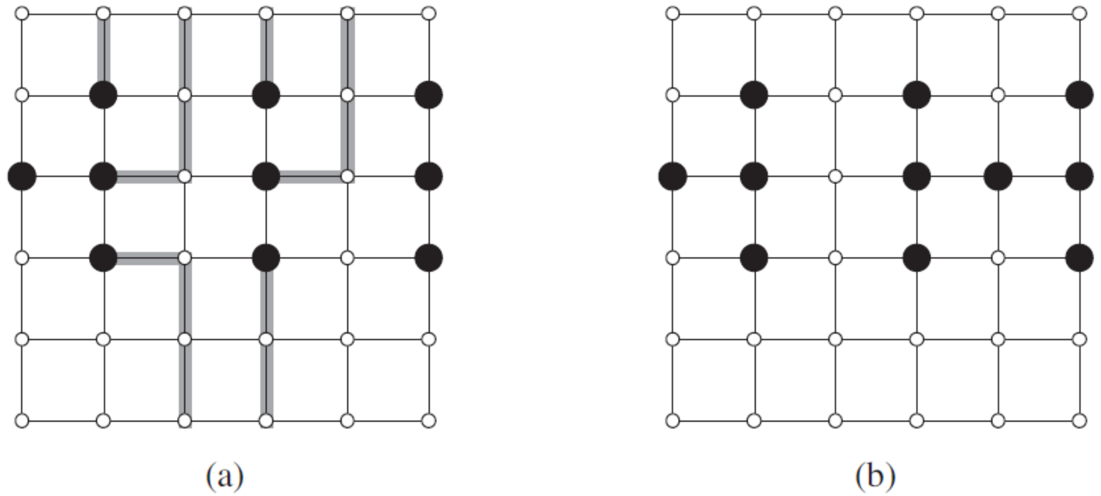
취구멍의 capacity=2



Escape problem

An $n \times n$ ***grid*** is an undirected graph consisting of n rows and n columns of vertices, as shown in Figure 26.11. We denote the vertex in the ith row and the jth column by (i, j). All vertices in a grid have exactly four neighbors, except for the boundary vertices, which are the points (i, j) for which $i = 1$, $i = n$, $j = 1$, or $j = n$.

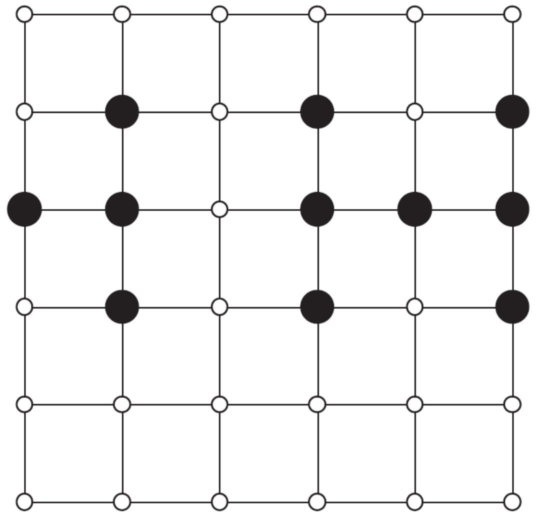
Given $m \leq n^2$ starting points $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$ in the grid, the ***escape problem*** is to determine whether or not there are m vertex-disjoint paths from the starting points to any m different points on the boundary. For example, the grid in Figure 26.11(a) has an escape, but the grid in Figure 26.11(b) does not.

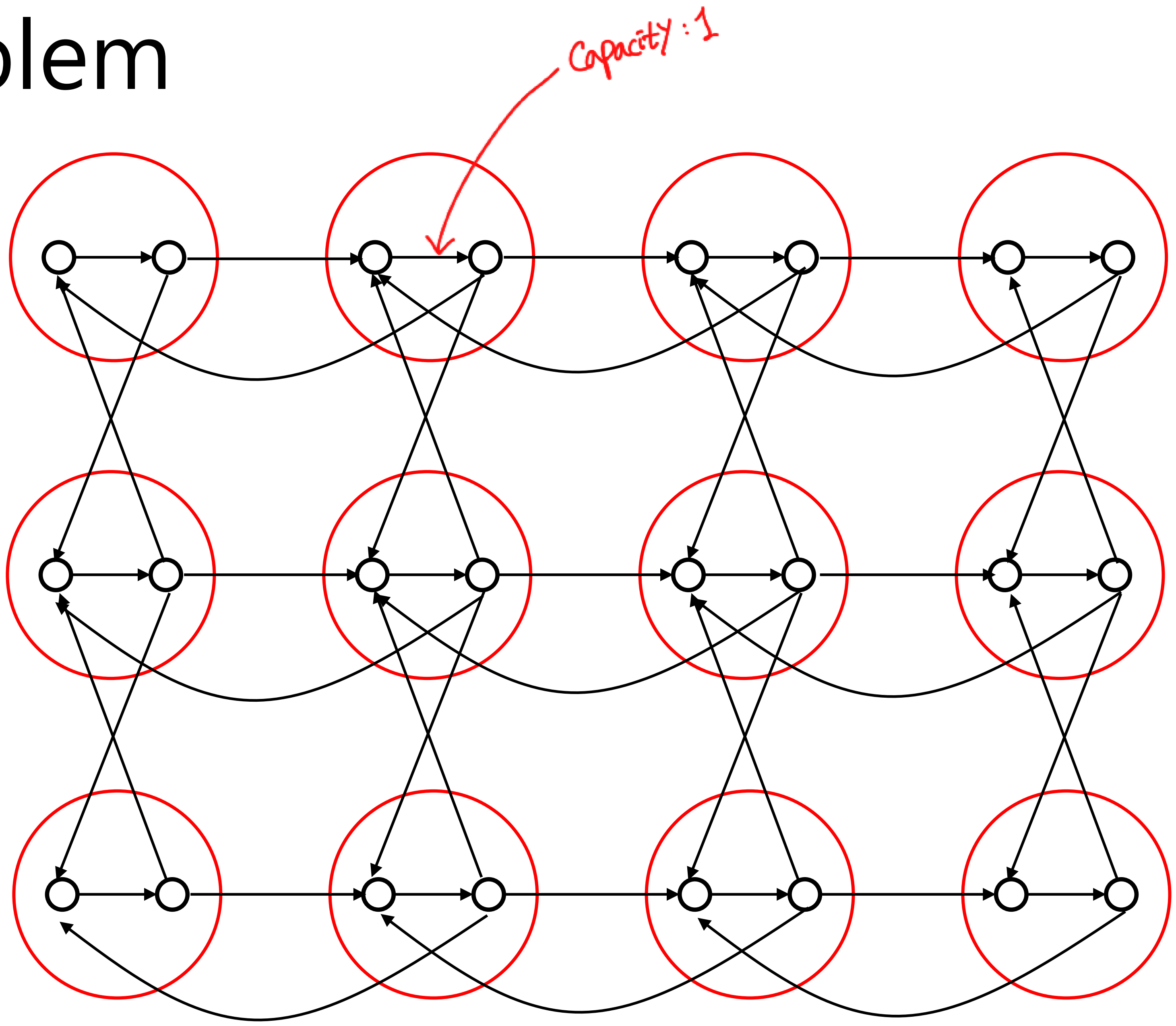


문제를 풀기 위한 모델링

- 이런 류의 문제를 풀기 위해선 문제에 적합하게 그래프를 설계해야 한다.
- 풀이를 스스로 찾을 때까지 이 곳에 머무릅시다.

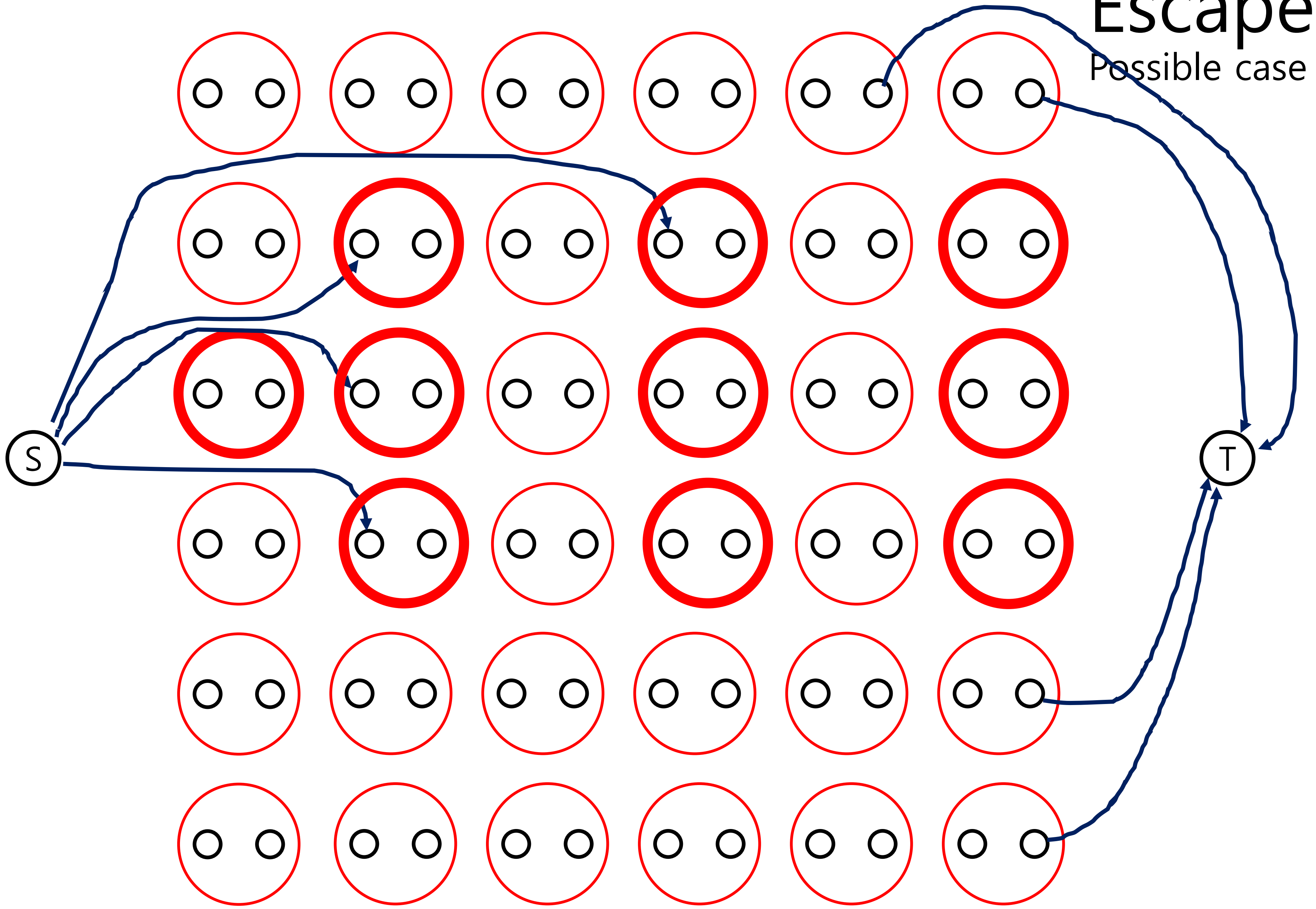
Escape problem

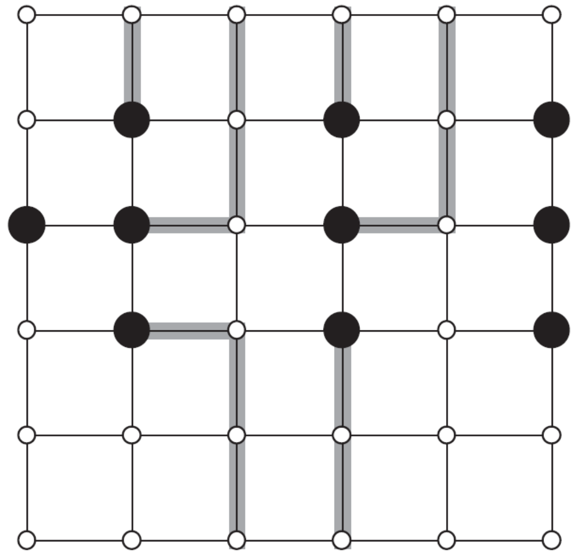




Escape problem

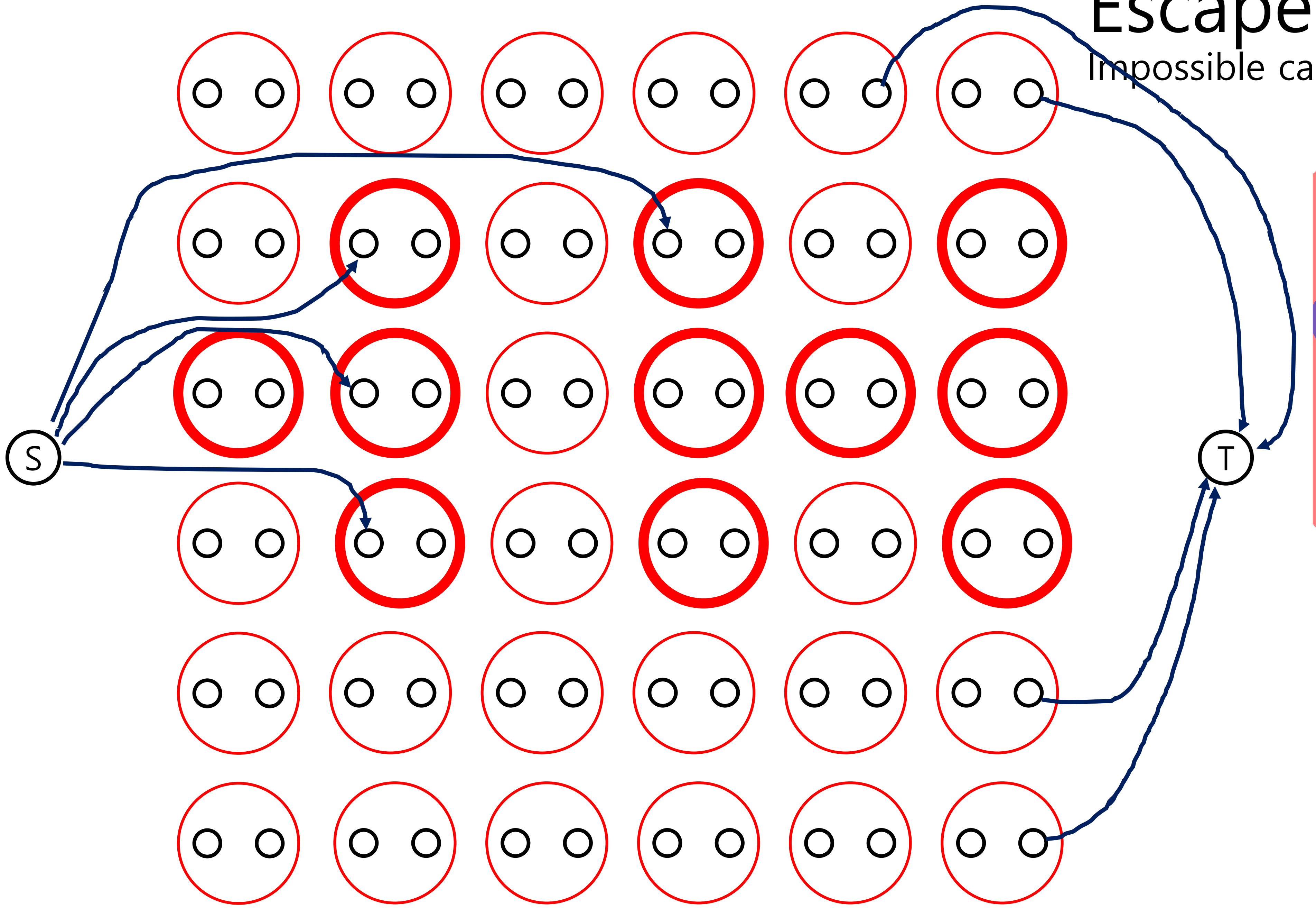
Possible case

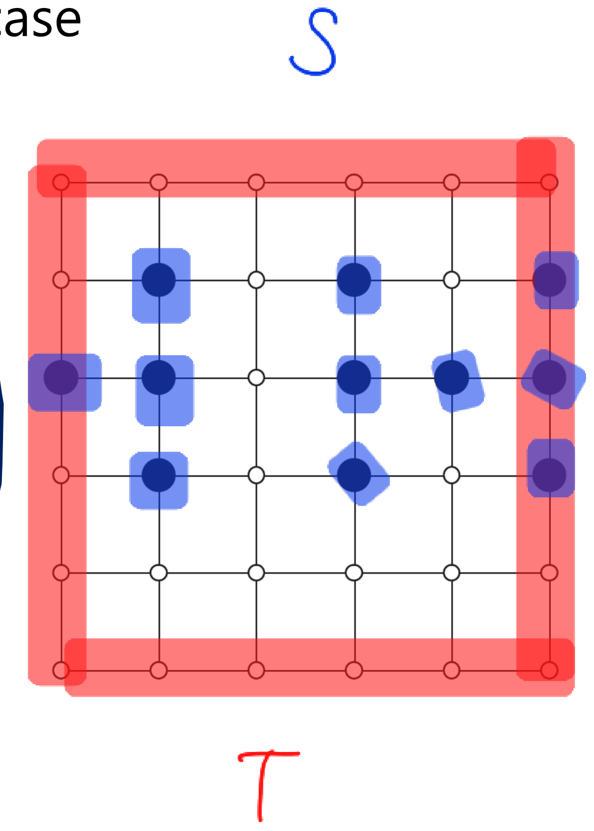




Escape problem

Impossible case





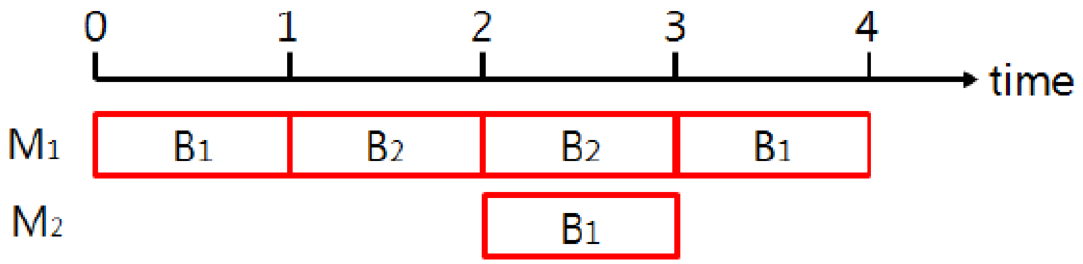
구두제작 문제

패션 구두 생산으로 유명한 회사인 A사는 고객의 주문에 따라 구두를 제작하며, 모든 구두는 기술이 뛰어난 K명의 장인에 의해 만들어진다. 주문을 받은 구두의 수를 N이라고 할 때, 각 구두 i $(1 \le i \le N)$ 에 대해 주문의뢰시간(arrival time) a_i와 제작이 완성되어야 하는 마감시간(finish time) f_i, 그리고 구두를 만들기 위해 소비해야 하는 총 제작시간(processing time) p_i가 주어진다 $(1 \le p_i \le f_i - a_i)$. 각 구두는 한 사람이 만들 수도 있고, 상황에 따라 여러 장인의 손을 거쳐 만들어지기도 한다. 그리고, 각 장인은 자기의 근무시간에만 구두를 만든다. 즉, 장인 j에 대해 근무시작시간(starting time) s_j, 근무마감시간(ending time) e_j가 주어진다 $(s_j < e_j)$. 어떤 특정 시각에, 한 장인이 둘 이상의 구두 제작에 참여할 수 없고 한 구두는 두 명 이상의 장인에 의해 제작되어지지 않는다. 즉, 각 장인은 한 순간에 한 구두만 만들며, 각 구두는 한 순간에 한 장인에 의해서만 제작된다. 모든 입력 값은 정수로 주어지며, 어떤 장인이 특정 구두 제작에 참여할 때도 항상 정수 시간만큼 소비한다고 가정한다.

장인 $[s_j, e_j]$ 구두 $[a_i, f_i, p_i]$

구두제작 문제

예를 들어, 장인이 두 명 (M_1, M_2) 이고, 각 장인의 근무시간 $[s_j, e_j]$이 각각 [0,4]와 [2,3]이다. 주문 받은 두 켤레의 구두 (B_1, B_2) 에 관한 정보 $[a_i, f_i, p_i]$는 각각 [0,4,3], [1,3,2]일 경우, 아래 그림은 각 장인이 각 구두에 어떻게 작업시간을 배당하는지를 나타내는 한 예이다.

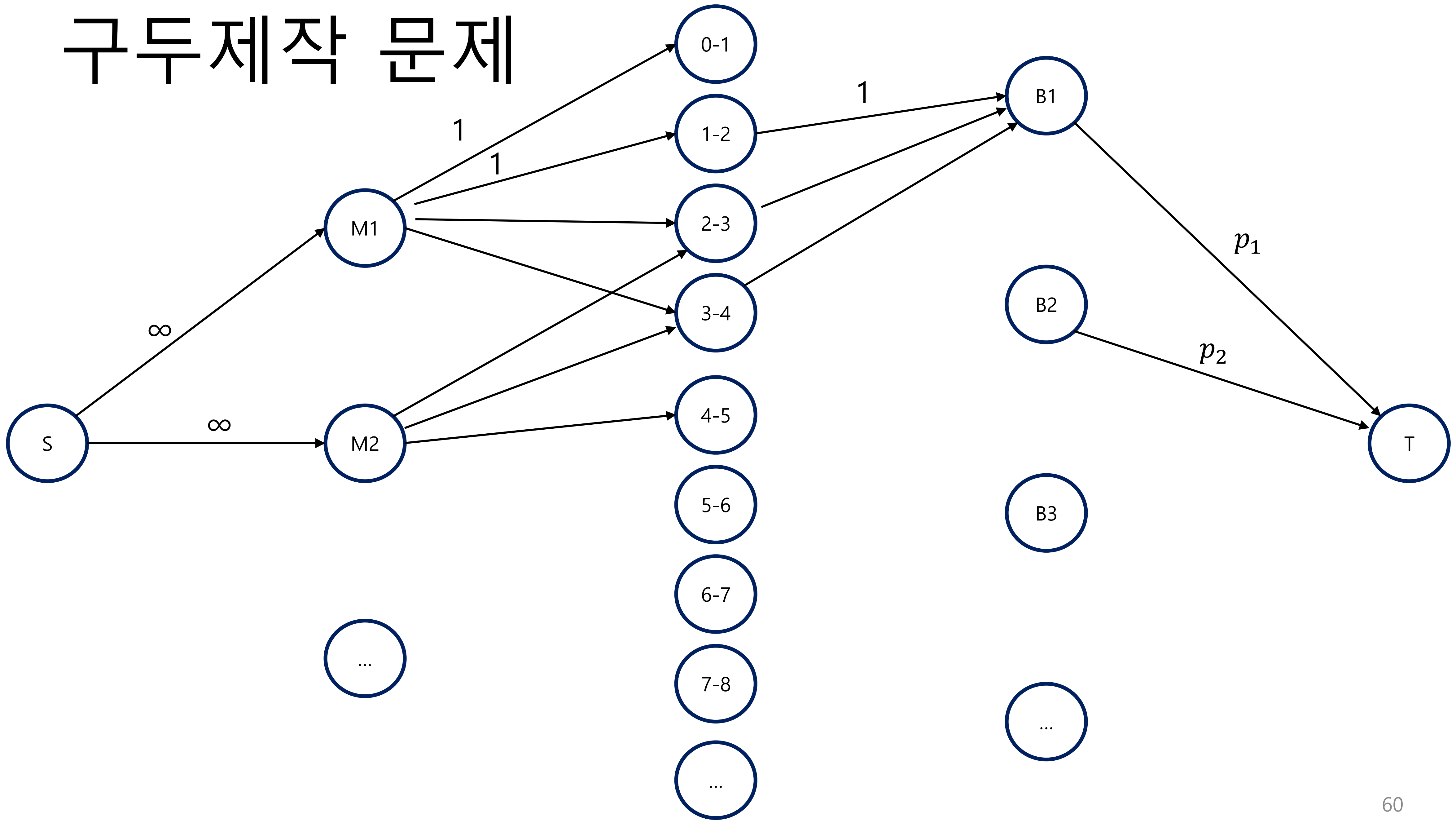


K명의 장인에 관한 정보와 N 켤레의 구두 정보가 주어질 때, 모든 구두를 제한시간 내에 장인들이 제작 가능한지 여부를 밝히는 프로그램을 작성하라.

구두제작 문제

- 문제를 위한 모델링을 어떻게?
- 각자 그래프를 만든 다음 풀이를 봅시다.

구두제작 문제



0-1
1-2
2-3
3-4
4-5
5-6
6-7
7-8
…
M1
M2
…
S
T
B1
B2
B3
…
1
1
1
∞
∞
p_1
p_2