Ford-Fulkerson Method
Max-Flow min-cut theorem

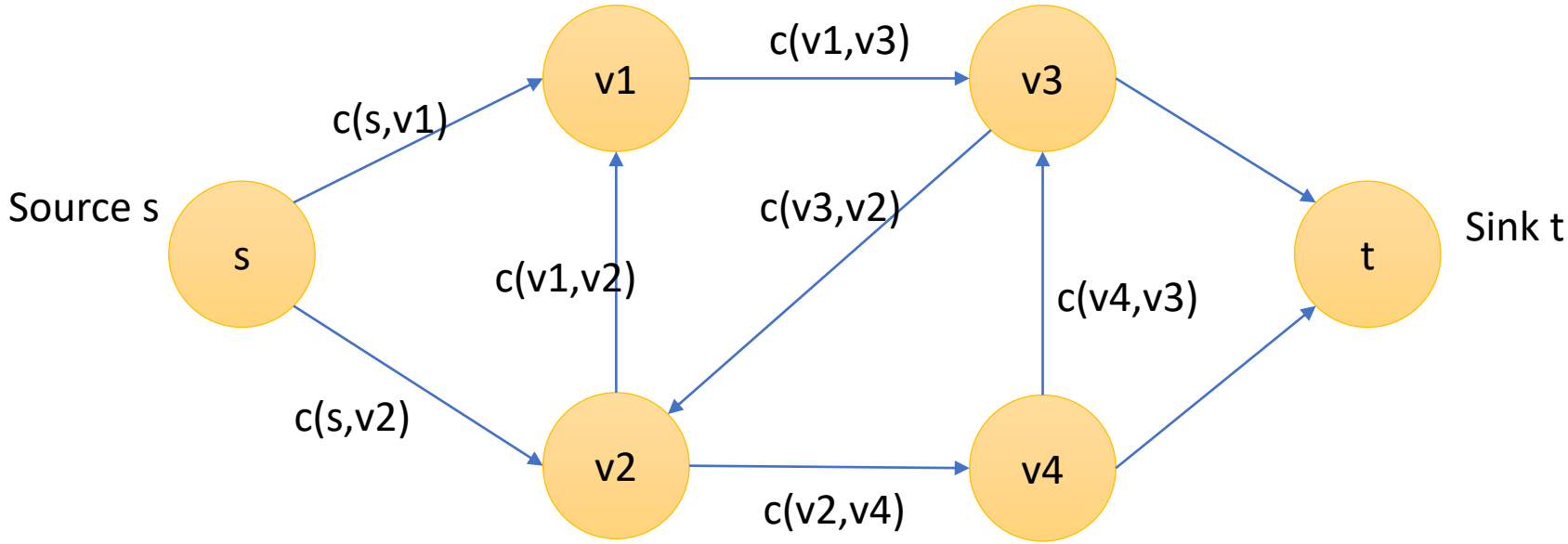
CS-6th

Instructor: Dr. Ayesha Enayet

List of Contents

Flow Graph G=(V,E)

- For each (u,v) ∈ E, c(u,v) is the flow capacity of each edge.



Note: Each edge could have a label of the form a/b where are presents the flow (f(u,v)) and b is capacity (c(u,v)).

Flow Networks

- A flow network G=(V,E) is a directed graph in which each edge (u,v) ∈ E has a nonnegative capacity c(u,v)>=0. We further require that if E contains an edge (u,v), then there is no edge (v,u) in the reverse direction.
- If (u,v) does not belongs to E, then for convenience we define c(u,v)=0, and we disallow self-loops.
- Each flow network contains two distinguished vertices: a source s and a sink t.
- For each vertex v ∈ V , the flow network contains a path s to v to t. Because each vertex other than s has at least one entering edge, we have E>=V -1.

Flow network: Properties

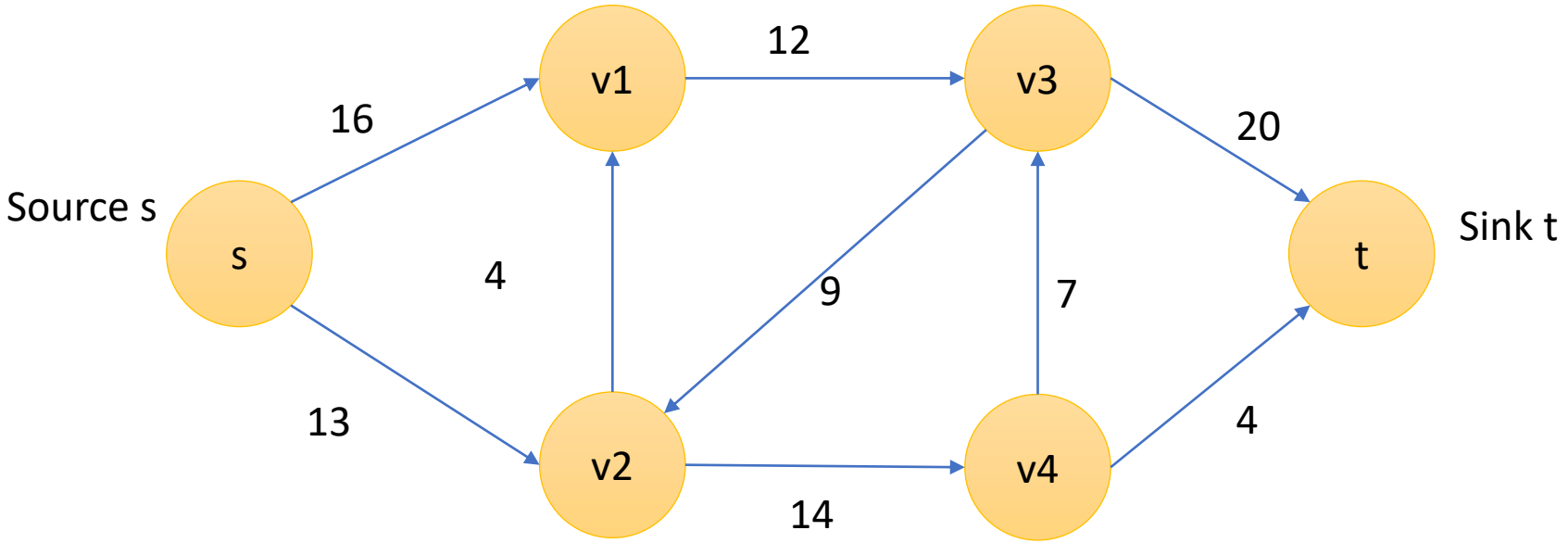
- Capacity Constraints:
  - For all (u,v) ∈ V, we require

$$0<=f(u,v)<=c(u,v)$$

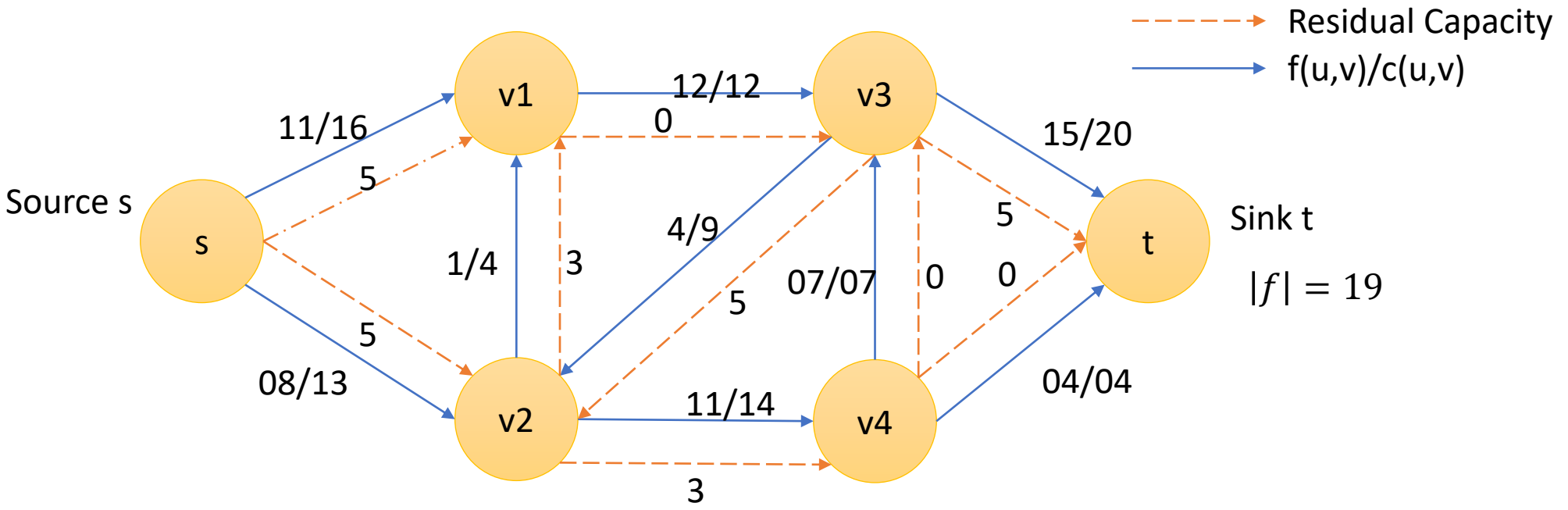
- Flow Conservation (in-flow=out-flow):
  - For all u ∈ V-{s,t}, we require

$$\sum_{u\in V} f(u,v) = \sum_{v\in V} f(v,u)$$

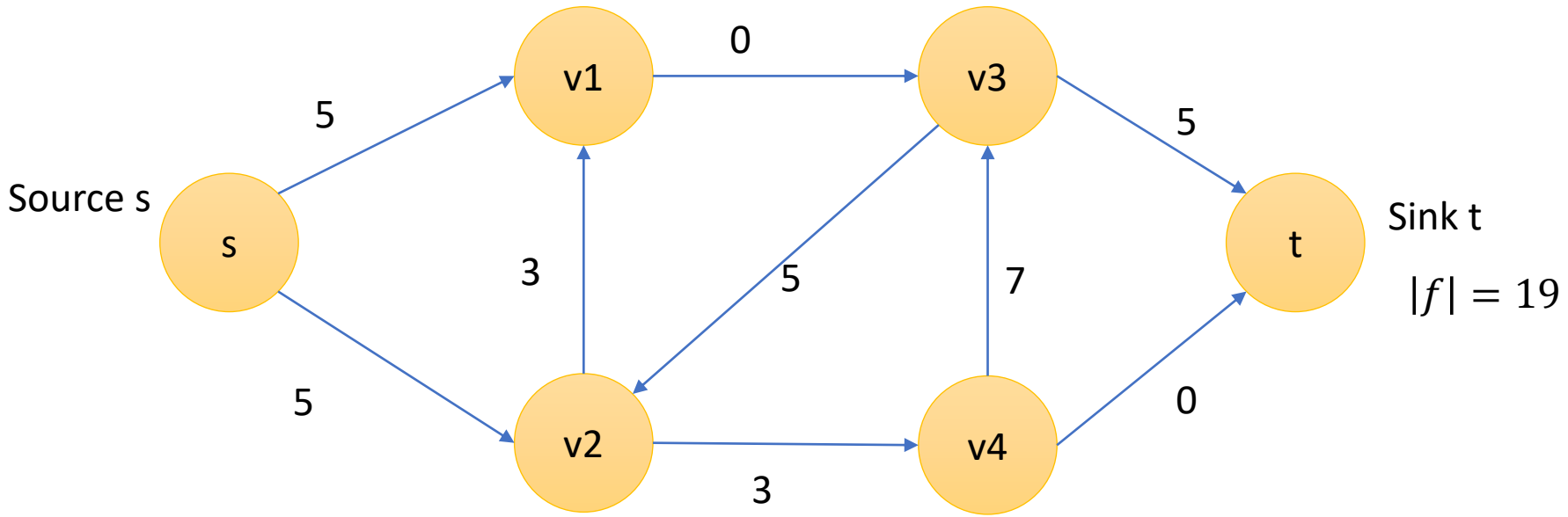
Flow Graph (Example)



Flow Graph with flow and residual capacity



Residual Graph (with only residual capacity)

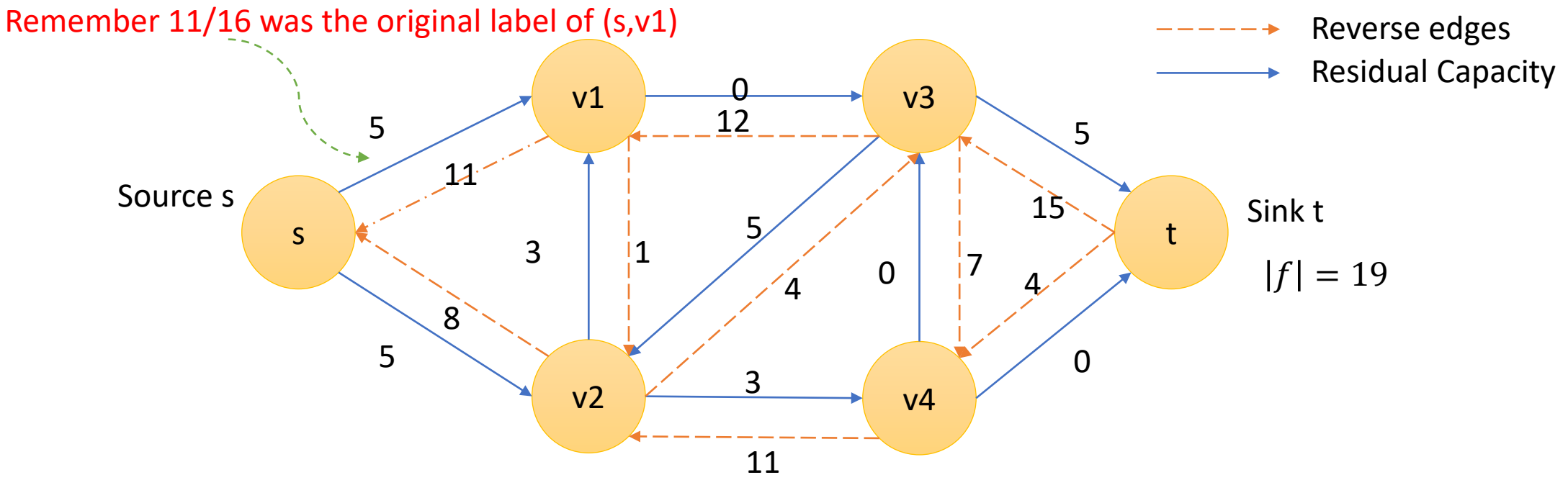


Note: Edges are marked with their residual capacity

Residual Capacity ($c_f\,(u, v)$)

- $c_f\,(u, v) = \begin{cases} c(u, v) - f(u, v) \;\; if (u, v) \in E, \\ f(v, u) \;\; if (v, u) \in E \\ 0 \;\; otherwise \end{cases}$

Residual Graph G_f



- Note: These reverse edges in the residual network allow an algorithm to send back flow it has already sent along an edge. Sending flow back along an edge is equivalent to decreasing the flow on the edge, which is a necessary operation in many algorithms. Unlike flow graphs, residual graphs can have both (u,v) and (v,u) edges.

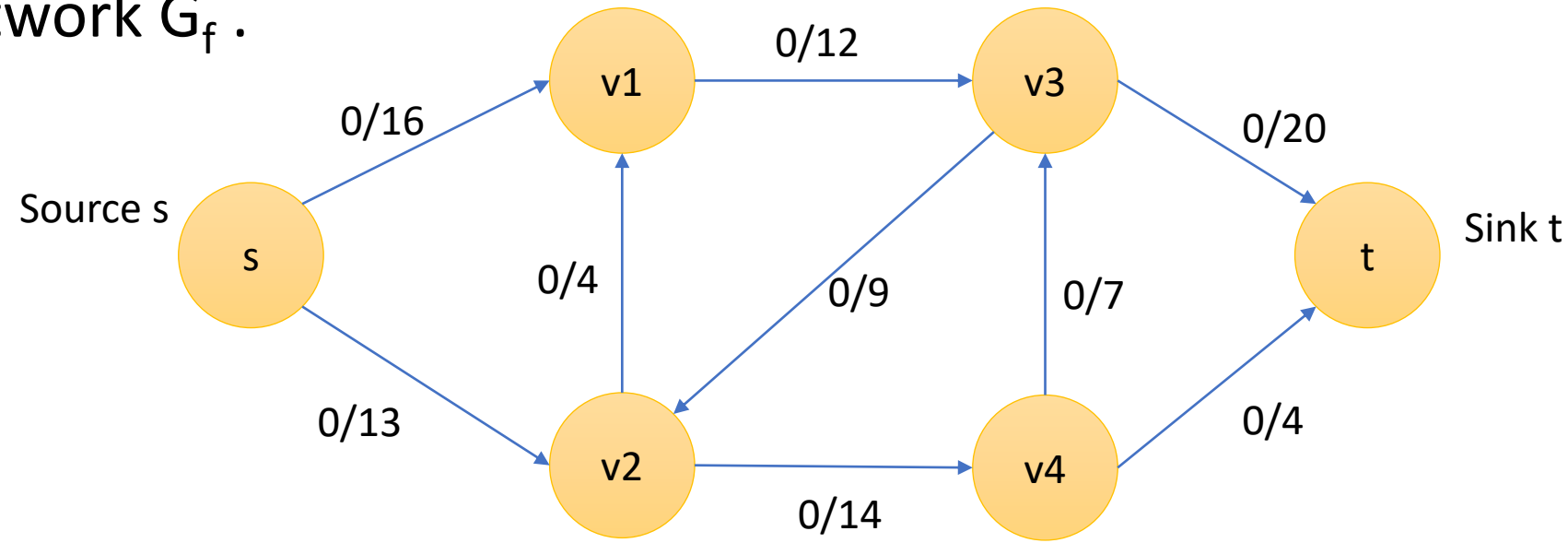
Ford-Fulkerson-Method

Finding a max flow is an optimization problem and Ford-Fulkerson-Method is used to find the max flow across the flow network/Graph.

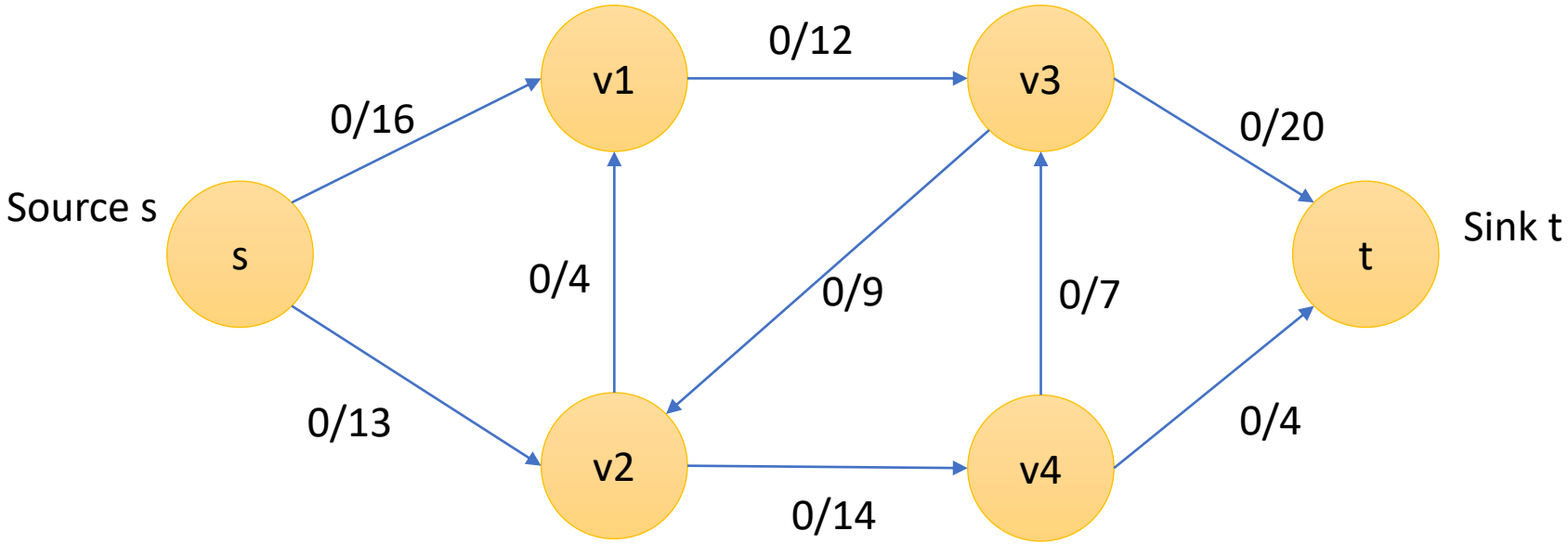
```
FORD-FULKERSON-METHOD(G, s, t)
1  initialize flow f to 0
2  while there exists an augmenting path p in the residual network G_f
3        augment flow f along p
4  return f
```

Initialization (Example)

- An augmenting path p is a simple path from s to t in the residual network G_f .



Residual Graph G_f with $|f| = 0$



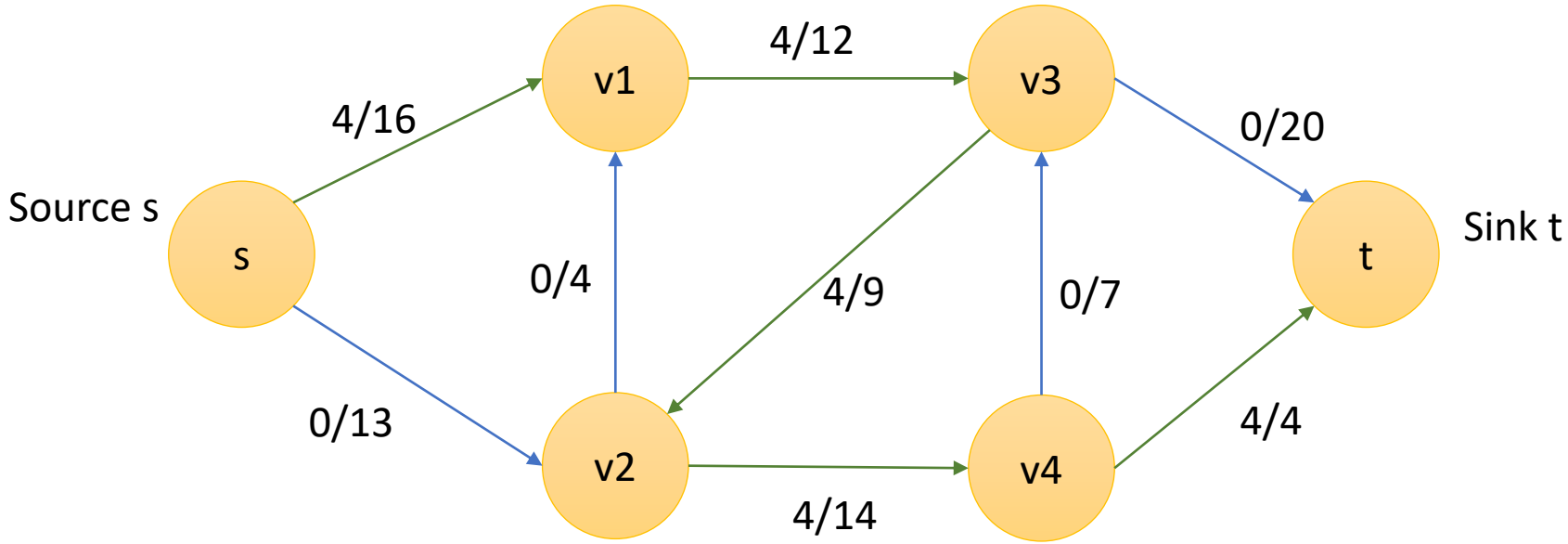
Augmenting path

- Given a flow network G=(V,E) and a flow f , an augmenting path p is a simple path from s to t in the residual network G_f . By the definition of the residual network, the flow on an edge (u,v) of an augmenting path may increase by up to $c_f(u,v)$ without violating the capacity constraint on whichever of (u,v) and (v,u) belongs to the original flow network G.
- We call the maximum amount by which we can increase the flow on each edge in an augmenting path p the residual capacity of p, given by

$$c_f(p) = \min\{c_f(u,v) : (u,v) \text{ is in } p\} \ .$$

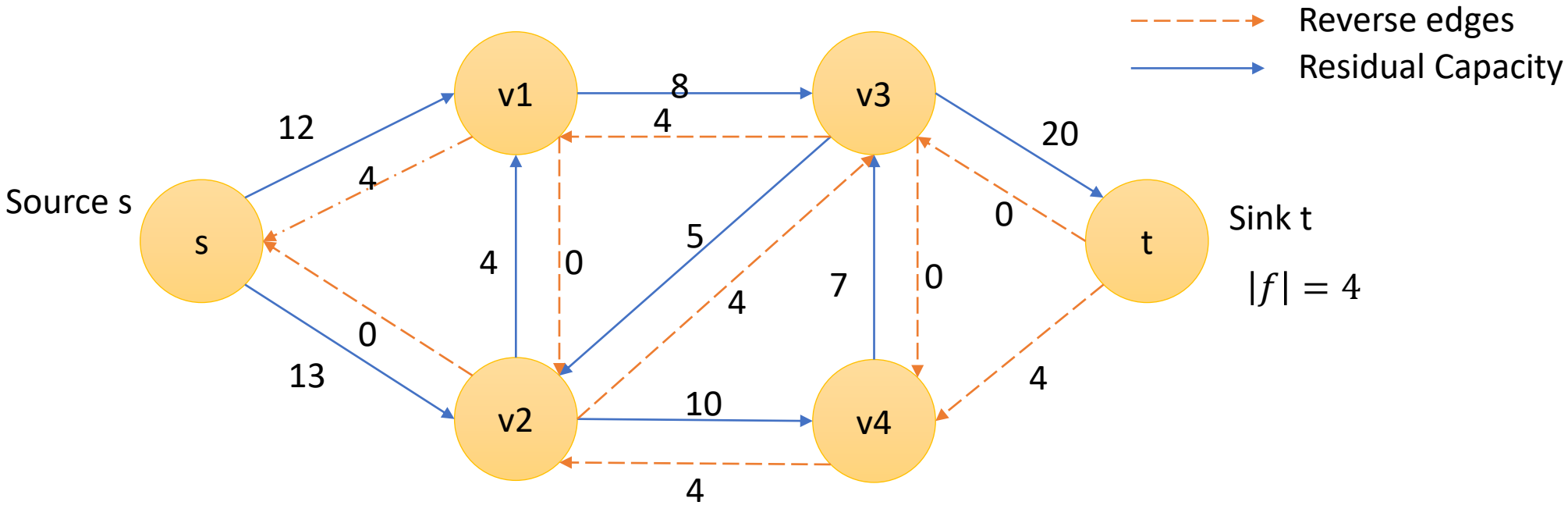
Flow Graph G with $|f| = 4$

- Consider an augmenting path s→v1→v3→v2→v4→t



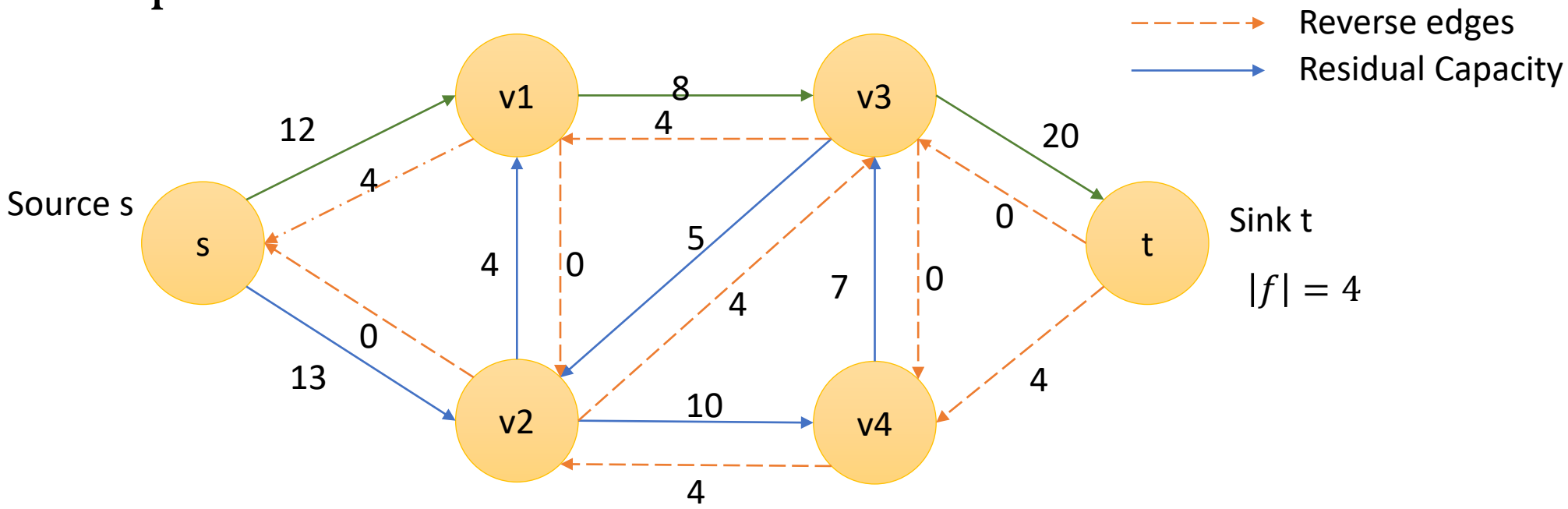
- Min(c(s,v1),c(v1,v3),c(v3,v2),c(v2,v4),c(v4,t))=4

Residual Graph G_f



Residual Graph G_f

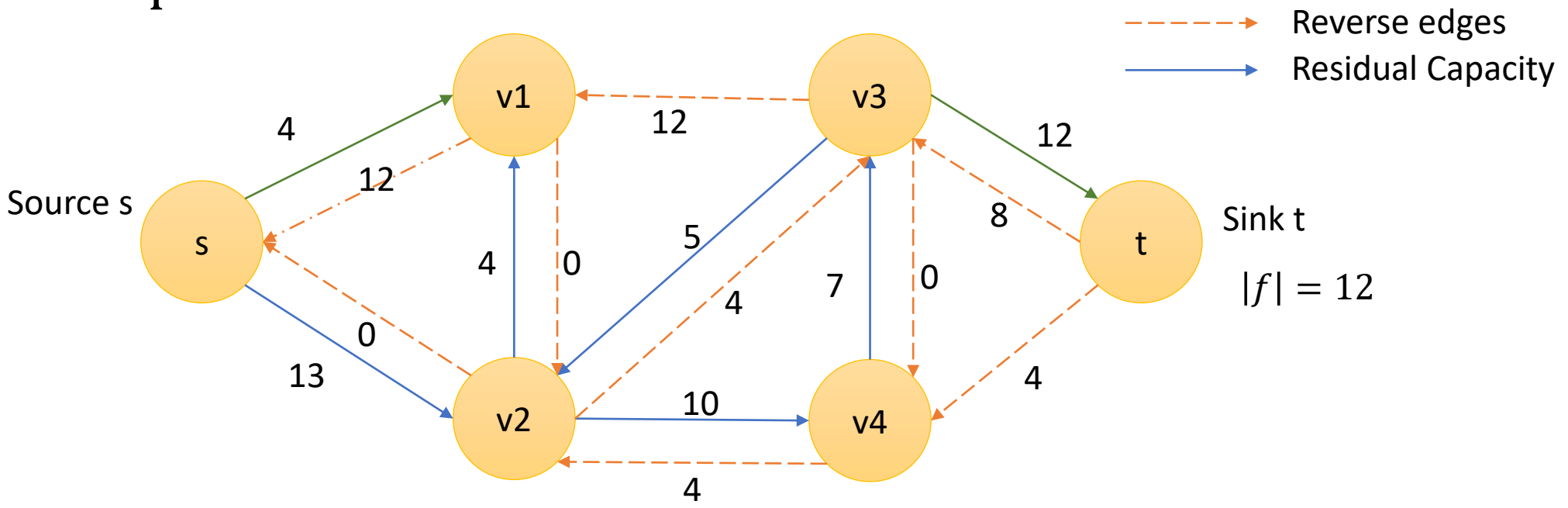
- Consider path s→v1→v3→t



- Min((s,v1),(v1,v3),(v3,t))=8

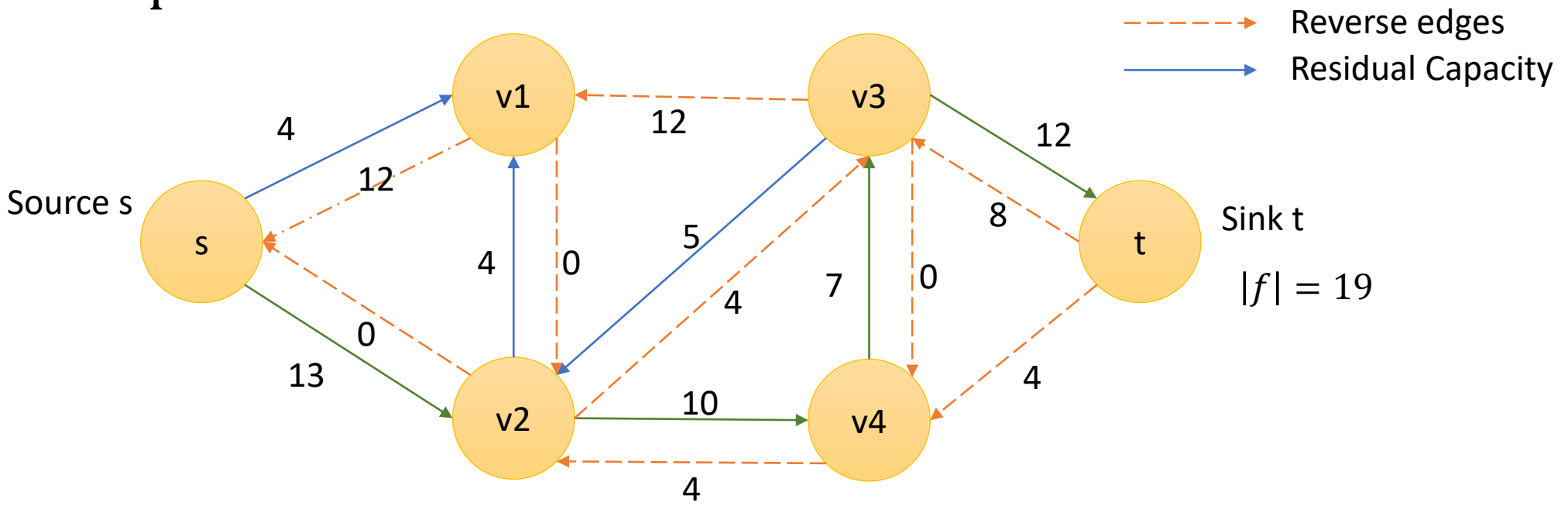
Residual Graph G_f

- Consider path s→v1→v3→t



Residual Graph G_f

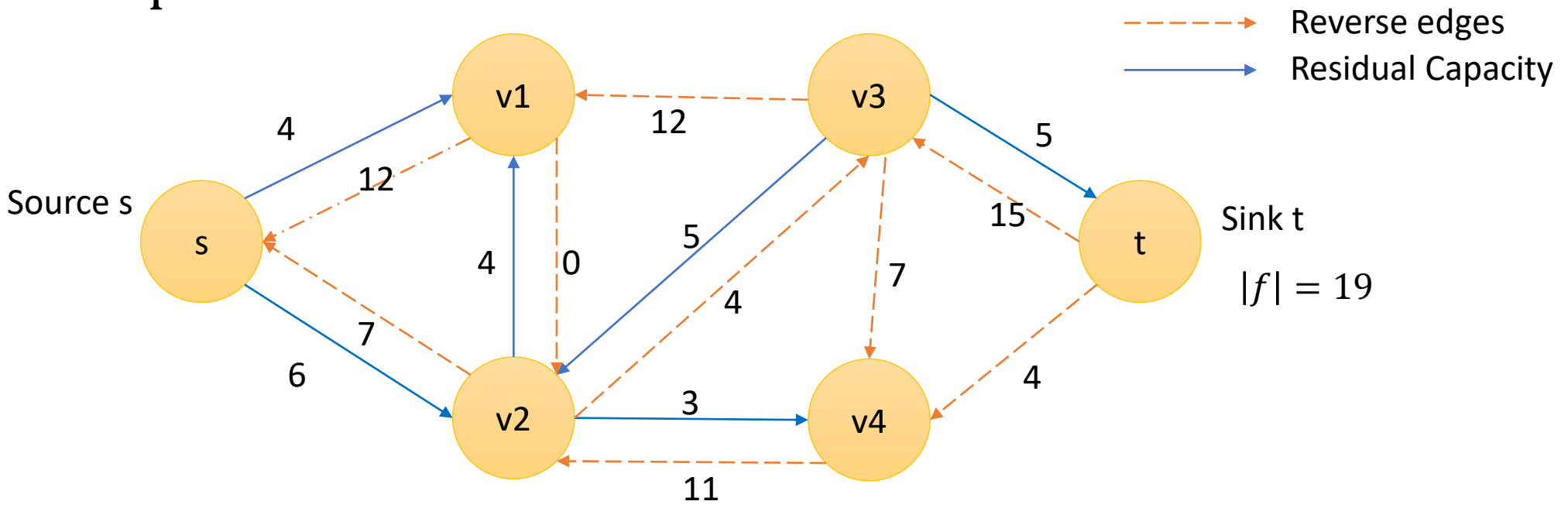
- Consider path s→v2→v4→v3→t



- Min((s,v2),(v2,v4),(v4,v3),(v3,t))=7

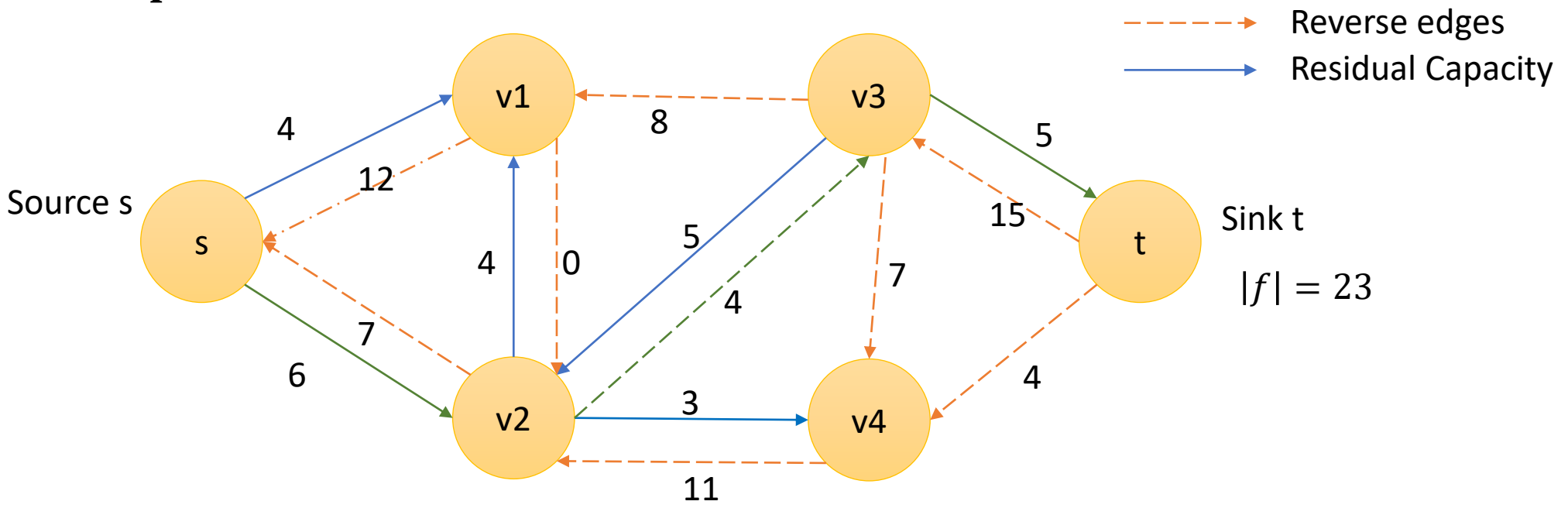
Residual Graph G_f

- Consider path s→v2→v4→v3→t



Residual Graph G_f

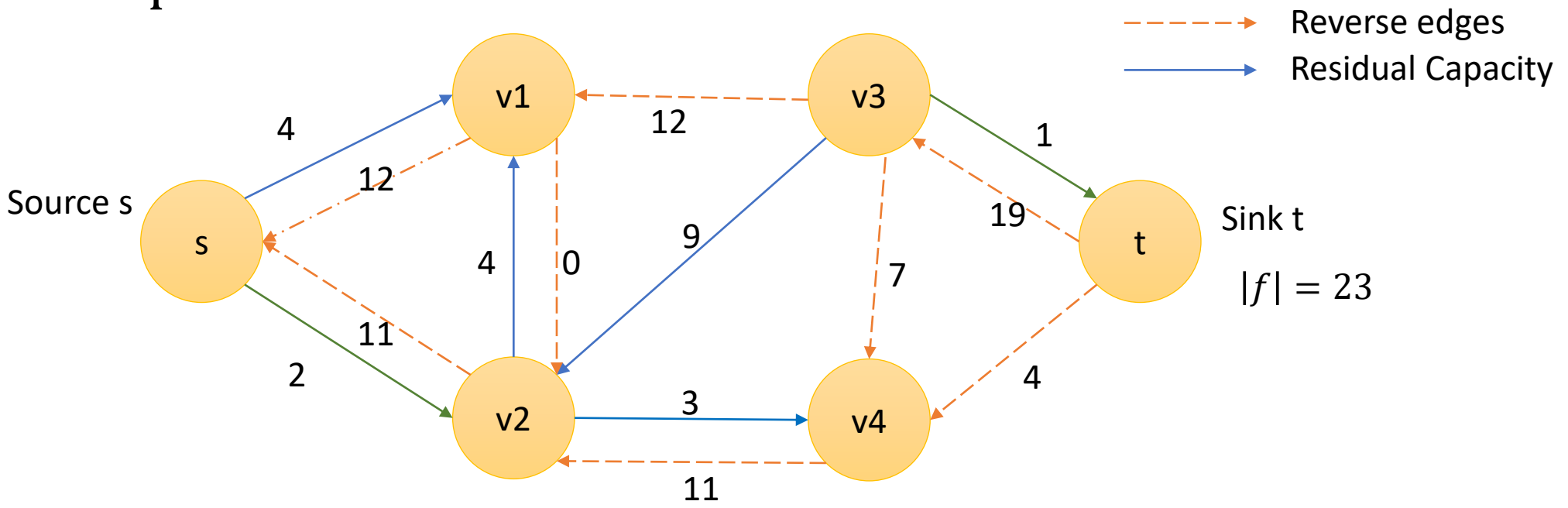
- Consider path s→v2→v3→t



- Min((s,v2),(v2,v3),(v3,t))=4

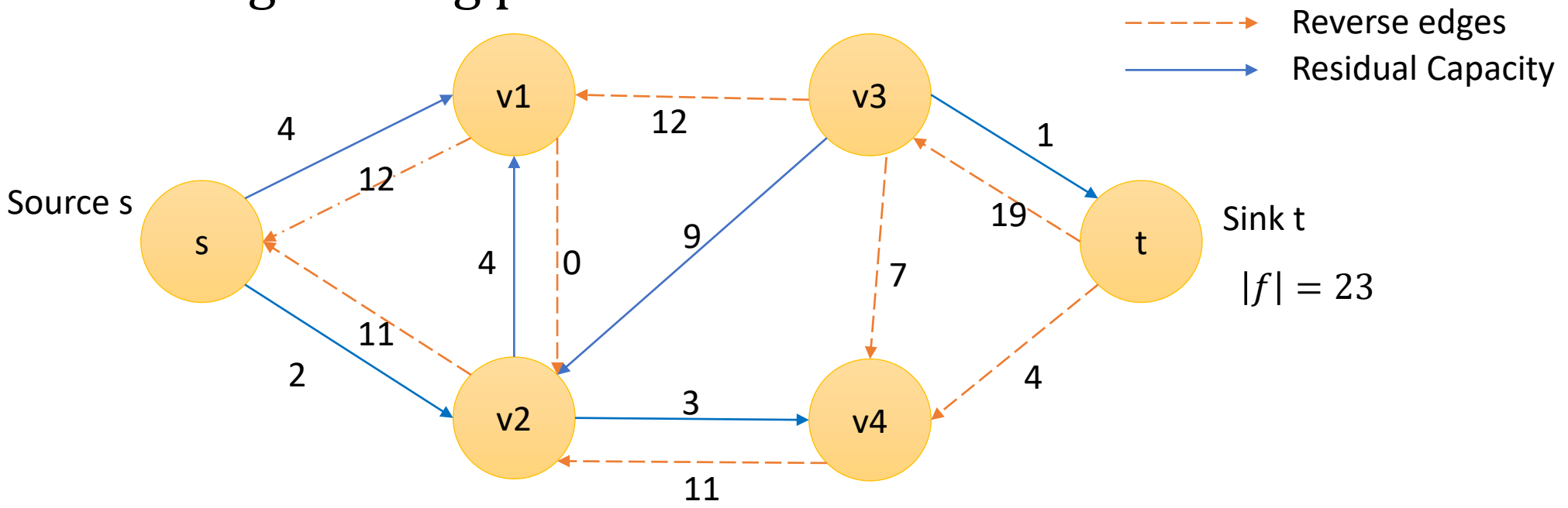
Residual Graph G_f

- Consider path s→v2→v3→t



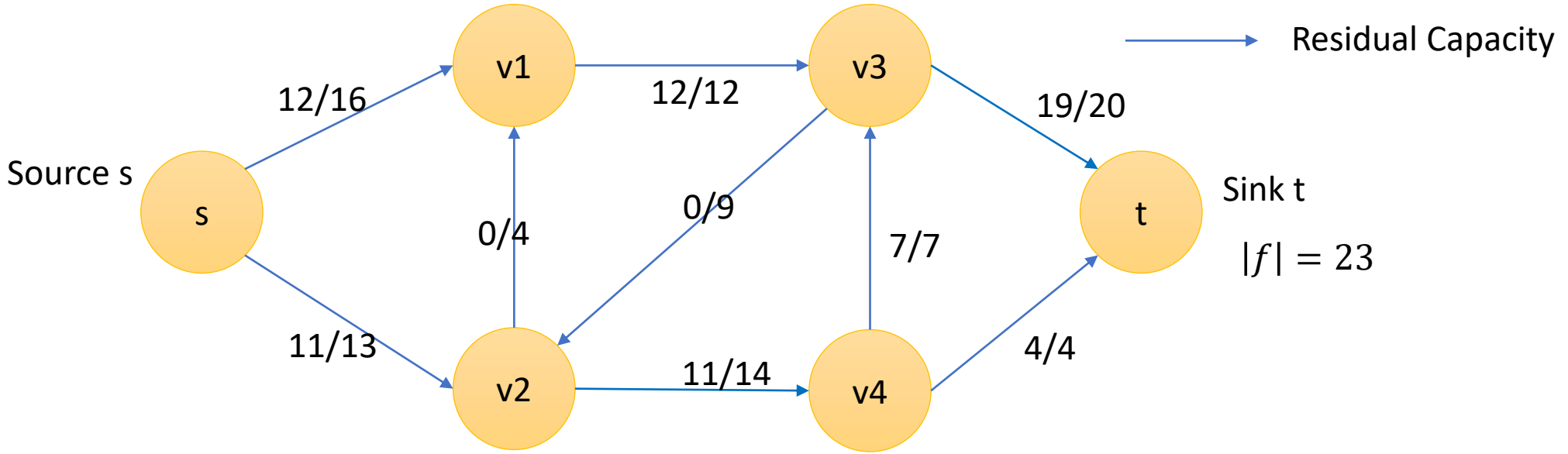
Residual Graph G_f

- No further augmenting path



Flow Graph

- Consider path s→v2→v3→t



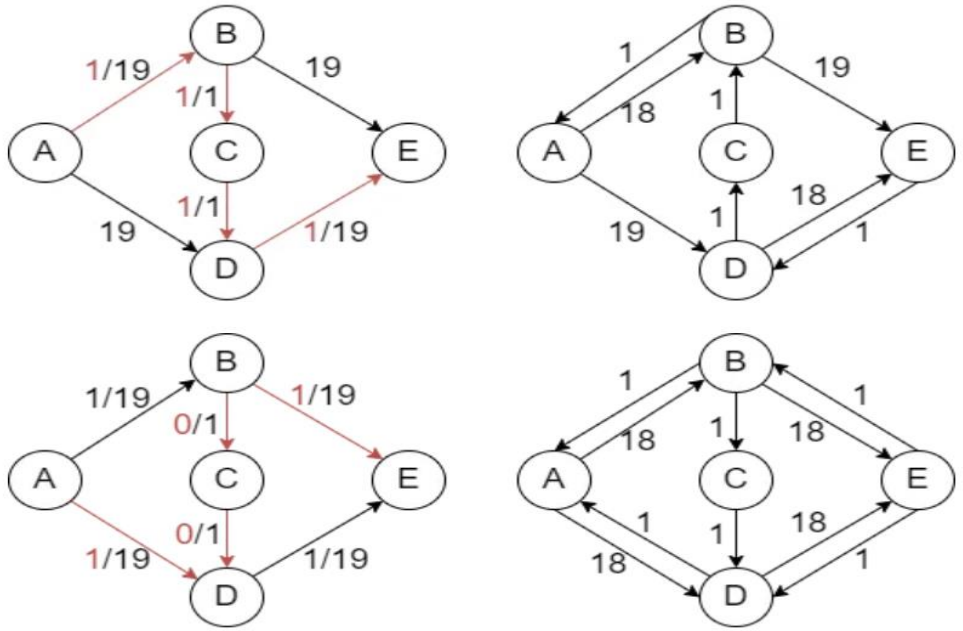
Ford-Fulkerson-Algorithm

```
FORD-FULKERSON(G, s, t)
1  for each edge (u, v) ∈ G.E
2      (u, v).f = 0
3  while there exists a path p from s to t in the residual network G_f
4      c_f(p) = min {c_f(u, v) : (u, v) is in p}
5      for each edge (u, v) in p
6          if (u, v) ∈ G.E
7              (u, v).f = (u, v).f + c_f(p)
8          else (v, u).f = (v, u).f − c_f(p)
9  return f
```

Does not specify a method to find augmenting path!

Time complexity (Worst case)

The worst-case scenario for it could be the following: first you find an augmenting path A-B-C-D-E, and then the path A-D-C-B-E.



As you can see, on each step you find an augmenting path with the flow of 11. You repeat these steps on and on, leading to f iterations in total.
Reference: https://hyperskill.org/learn/step/27203

Time complexity

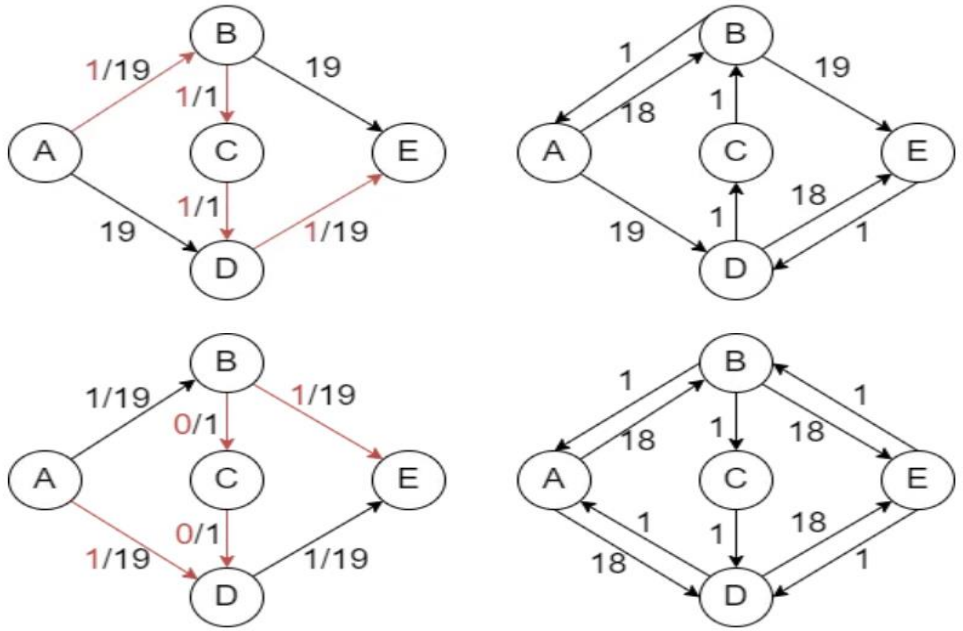
- If |f| denotes a maximum flow in the transformed network, then a straightforward implementation of FORD-FULKERSON executes the while loop of lines 3-8 at most |f| times, since the flow value increases by at least 1 unit in each iteration.
- The Ford-Fulkerson method using DFS to find augmenting paths takes O(Ef) where E is the number of edges and f is the max flow.
- DFS could lead to a longer path, which increases the chance of encountering a smaller bottleneck value and results in a longer run time.

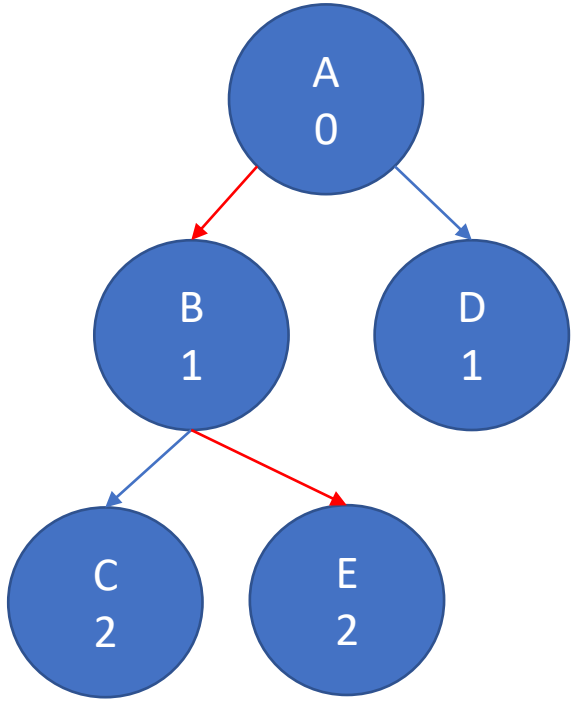
Time Complexity

- A good implementation should perform the work done within the while loop efficiently. It should represent the flow network G=(V,E) with the right data structure and find an augmenting path by a linear-time algorithm.

Lets try BFS!

The worst-case scenario for it could be the following: first you find an augmenting path A-B-C-D-E, and then the path A-D-C-B-E.





Reference: https://hyperskill.org/learn/step/27203

We found a shortest path A→B→E that we can augment to 19!

Time Complexity

- Edmonds-Karp algorithm:
  - By using breadth-first search to find an augmenting path in the residual network, the algorithm runs in polynomial time $O(VE^2)$, independent of the maximum flow value. We call the Ford-Fulkerson method so implemented the Edmonds-Karp algorithm.
  - **Edmonds-Karp** algorithm an implementation of Ford-Fulkerson method.
  - This algorithm is basically the same as the Ford-Fulkerson one. There is only one difference: in the Edmonds-Karp algorithm, you push the flow through the augmenting path that is the shortest and not just through any one. In this case, the shortest means consisting of the smallest number of edges.

Edmonds-Karp algorithm

The algorithm is identical to the Ford–Fulkerson algorithm, except that the search order when finding the augmenting path is defined. The path found must be a shortest path that has available capacity. This can be found by a breadth-first search, where we apply a weight of 1 to each edge. The running time of $O(V.E^2)$ is found by showing that each augmenting path can be found in $O(E)$ time, that every time at least one of the Edges E becomes saturated, that the distance from the saturated edge to the source along the augmenting path must be longer than last time it was saturated, and the length is at most $|V|$.

Reference: Edmonds–Karp algorithm - Wikipedia

Max-Flow min-cut theorem (correctness of Ford-Fulkerson method)

CS-6th

Instructor: Dr. Ayesha Enayet

Cuts of flow networks

- A cut (S,T) of flow network G(V,E) is a partition of V into S and T=V-S such that s ∈ S and t ∈ T (disjoint sets).
- If f is a flow, then the net flow f(S,T) across the cut (S,T) is defined to be

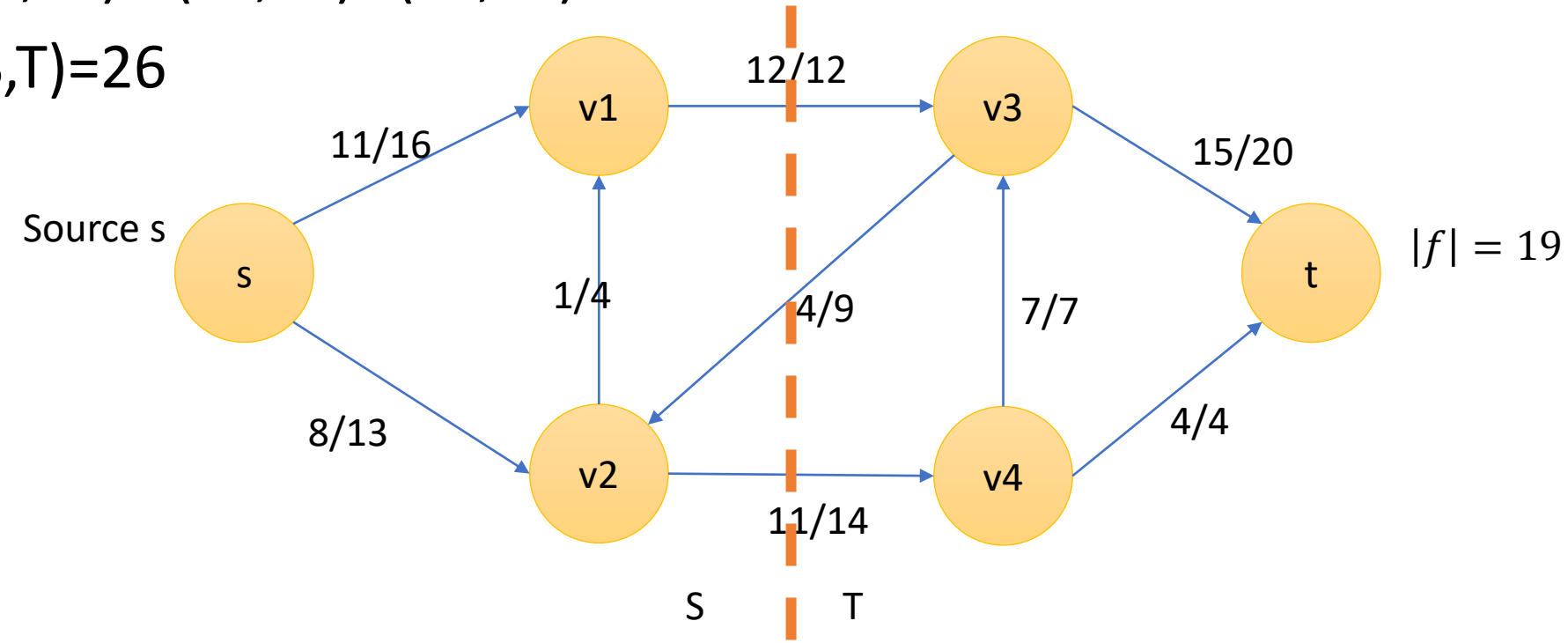
$$f(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v) - \sum_{u \in S} \sum_{v \in T} f(v,u) \ .$$

The ***capacity*** of the cut (S,T) is

$$c(S,T) = \sum_{u \in S} \sum_{v \in T} c(u,v) \ .$$

Cuts of flow networks (Example)

- f(v1,v3)+f(v2,v4)-f(v3,v2)=12+11-4=19
- C(S,T)=26



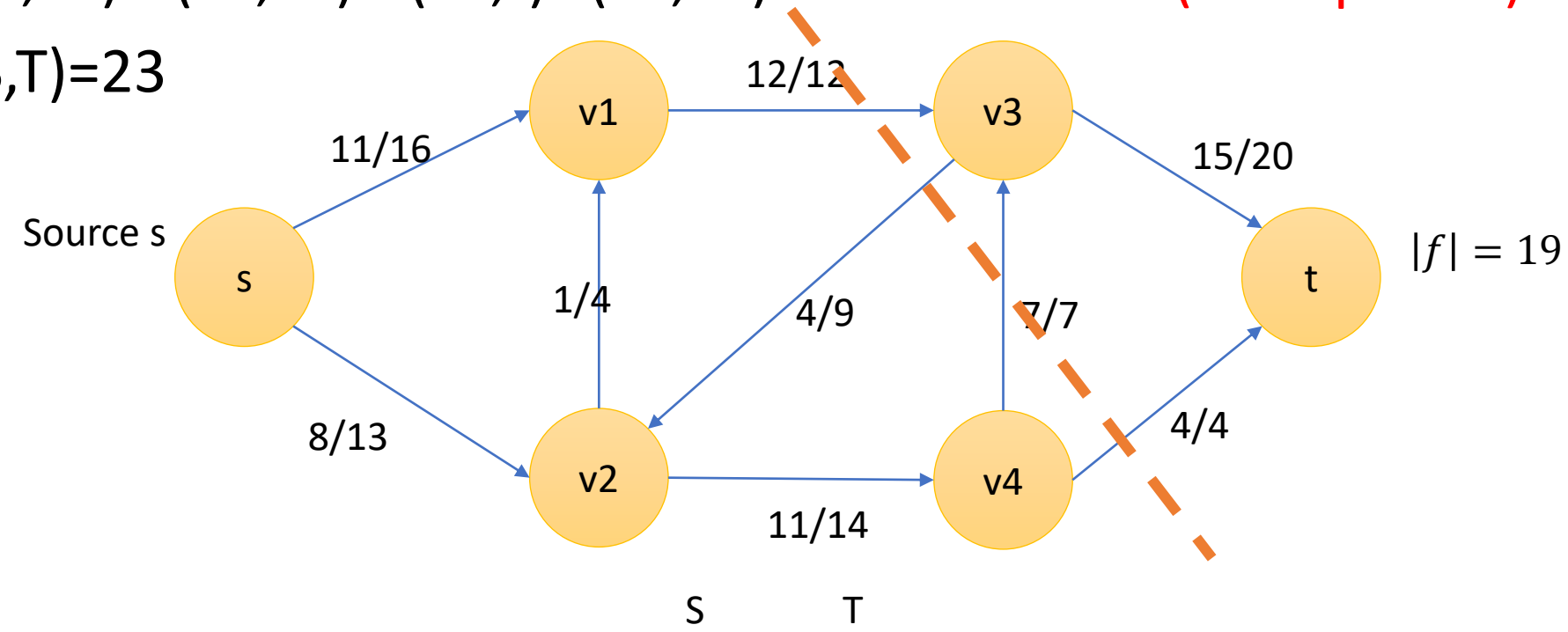
Is this a min-cut? No!

Min-cut

- A minimum cut of a network is a cut whose capacity is minimum over all cuts of the network.

Min-cut (Example)

- f(v1,v3)+f(v4,v3)+f(v4,t)-f(v3,v2)=12+7+4-4=19 (not optimal)
- C(S,T)=23



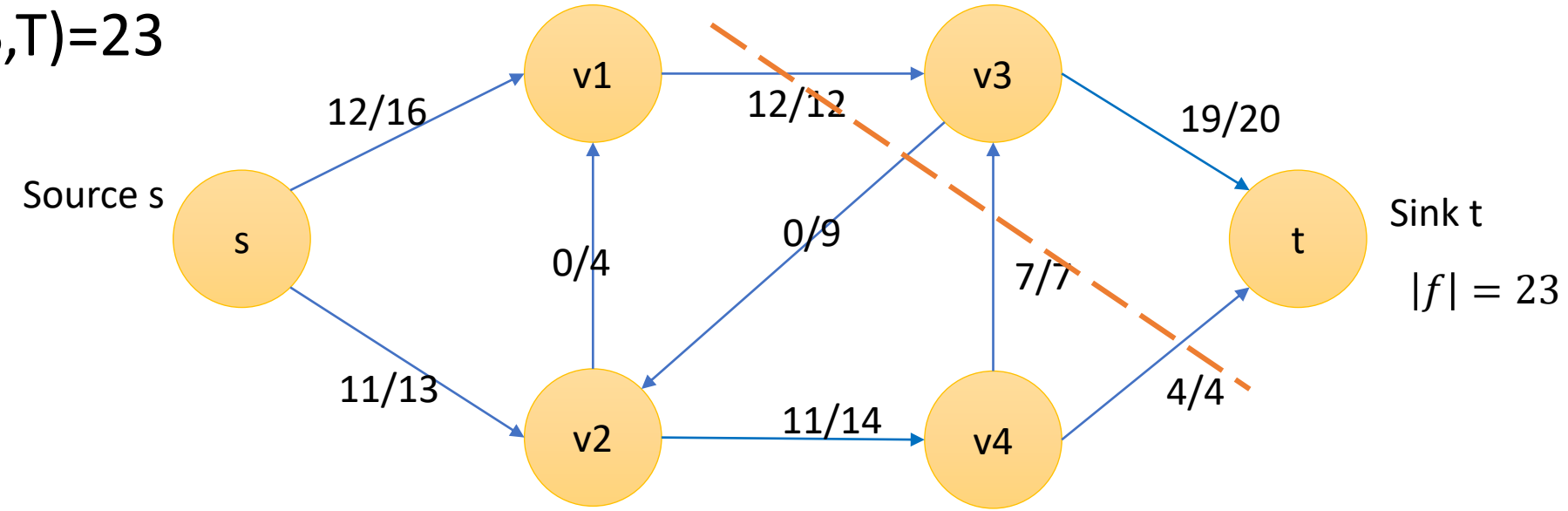
Is this a min cut? Yes!

How to check the correctness of the solution (max flow)?

By min-cut max-flow theorem which says “The maximum value of an s-t flow is equal to the minimum capacity over all s-t cuts”

Max-flow min-cut theorem

- f(v1,v3)+f(v4,v3)+f(v4,t)-f(v3,v2)=12+7+4-0=23
- C(S,T)=23



Max-flow min-cut theorem

- Lemma <>

Let f be a flow in a flow network G with source s and sink t, and let (S,T) be any cut of G. Then the net flow across (S,T) is f(S,T)= $|f|$.

- Corollary <>

The value of any flow f in a flow network G is bounded from above by the capacity of any cut of G.

Max-flow min-cut theorem

Theorem 24.6 (Max-flow min-cut theorem)
If f is a flow in 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.

Proof (1) $\Rightarrow$ (2): Suppose for the sake of contradiction that f is a maximum flow in G but that G_f has an augmenting path p. Then, by Corollary 24.3, the flow found by augmenting f by f_p, where f_p is given by equation (24.7), is a flow in G with value strictly greater than $|f|$, contradicting the assumption that f is a maximum flow.

(2) $\Rightarrow$ (3): Suppose that G_f has no augmenting path, that is, that G_f contains no path from s to t. Define

$$S = \{v \in V : \text{there exists a path from } s \text{ to } v \text{ in } G_f\}$$

and $T = V - S$. The partition (S, T) is a cut: we have $s \in S$ trivially and $t \notin S$ because there is no path from s to t in G_f. Now consider a pair of vertices $u \in S$ and $v \in T$. If $(u, v) \in E$, we must have $f(u, v) = c(u, v)$, since otherwise $(u, v) \in E_f$, which would place v in set S. If $(v, u) \in E$, we must have $f(v, u) = 0$, because otherwise $c_f(u, v) = f(v, u)$ would be positive and we would have $(u, v) \in E_f$, which again would place v in S. Of course, if neither (u, v) nor (v, u) belongs to E, then $f(u, v) = f(v, u) = 0$. We thus have

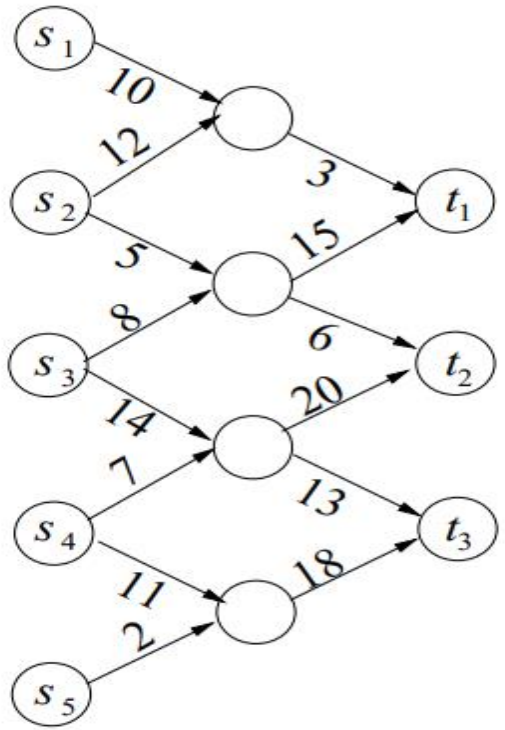
$$\begin{aligned} f(S, T) &= \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{v \in T} \sum_{u \in S} f(v, u) \\ &= \sum_{u \in S} \sum_{v \in T} c(u, v) - \sum_{v \in T} \sum_{u \in S} 0 \\ &= c(S, T) . \end{aligned}$$

By Lemma 24.4, therefore, $|f| = f(S, T) = c(S, T)$.

(3) $\Rightarrow$ (1): By Corollary 24.5, $|f| \leq c(S, T)$ for all cuts (S, T). The condition $|f| = c(S, T)$ thus implies that f is a maximum flow. ■

A multi-source multi-sink problem and its equivalent single-source single-sink version.

Multiple source, Multiple Sinks (Example)



Mapping to Single-Source, Single Sink