Introduction to Algorithms

Meng-Tsung Tsai

12/05/2019

Announcements

Programming Assignment 3 is due by Dec 27, 23:59. at https://oj.nctu.me
You may receive a bonus to hand in your assignment by Nov 20, 23:59.

Programming Quiz 2 will be held on Dec 28 (Sat), 13:30 - 17:30 at EC 315, 316, 324.

Written Assignment 3 is due by Dec 24, 10:20. at https://e3new.nctu.edu.tw It will be announced tomorrow evening.

Quiz 2 will be held on Dec 31, 10:10 - 11:00.

Scope of Programming Quiz 2

There are 5 problem sets and you may bring codes/slides/ebooks (electronic copies) with you using a USB flash drive and/or physical books/cheeting sheets.

The total size of e-files cannot exceed 200 MB, the number of physical books is at most 2, and the number of cheeting sheets is at most 4.

1. (60%) An application of BFS -- A Yes/No problem.
2. (20%) An application of Network Flows. You will be instructed how to use Ford-Fulkerson algorithm to solve this problem.
3. (15%) An application of Minimum Spanning Trees.
4. (15%) A challening problem. (Shortest Paths/SCC/T-Sort)
5. (15%) A more challenging problem. (Graph Problems)

It is hard to get fewer than 30 points. Please attend this quiz.

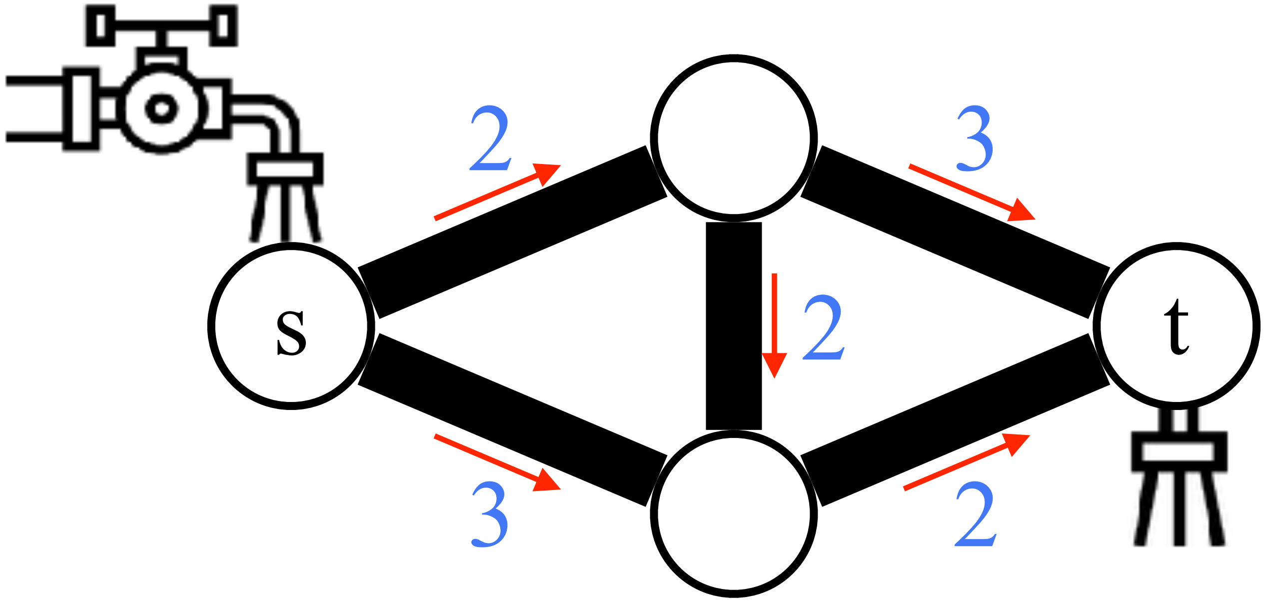
Network Flow

Problem

Input: a directed graph G and two distinguished nodes in G, which are a source node **s** and a sink node **t**. Every edge (u, v) in G has a nonnegative capacity c(u, v).

Output: the maximum flow from **s** to **t**.

Example.



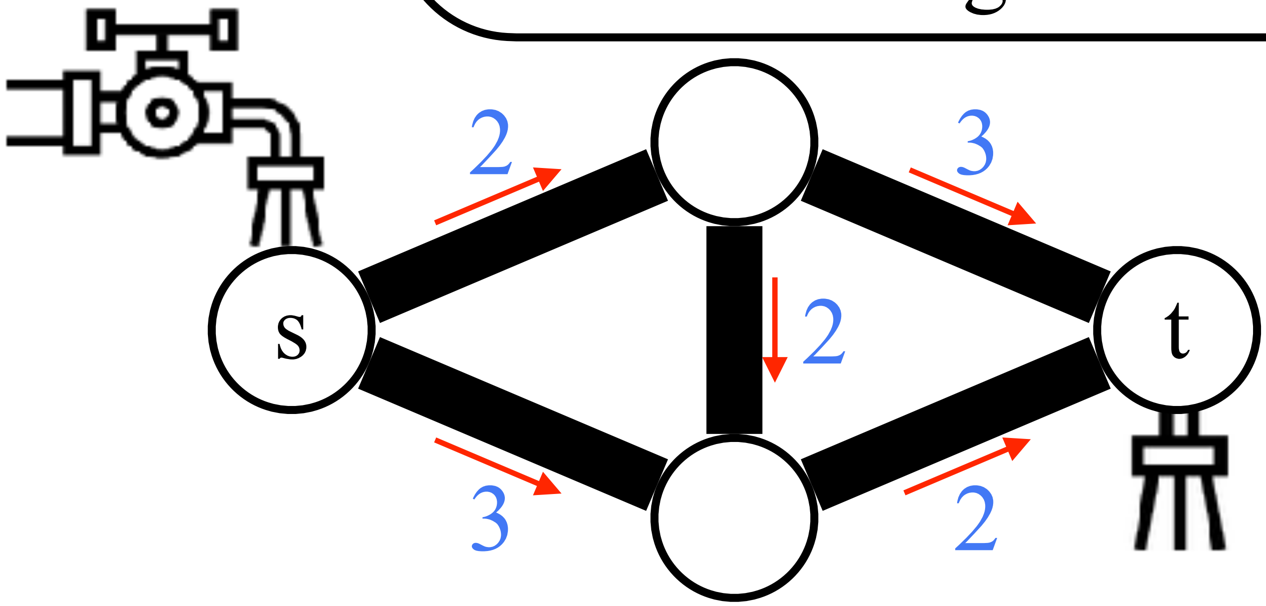
Problem

Input: a directed graph G and two distinguished nodes in G, which are a source node **s** and a sink node **t**. Every edge (u, v) in G has a nonnegative capacity c(u, v).

Output: the maximum flow from **s** to **t**.

Example.

c(u, v) is the maximum rate that water can flow through the directed pipe (u, v).



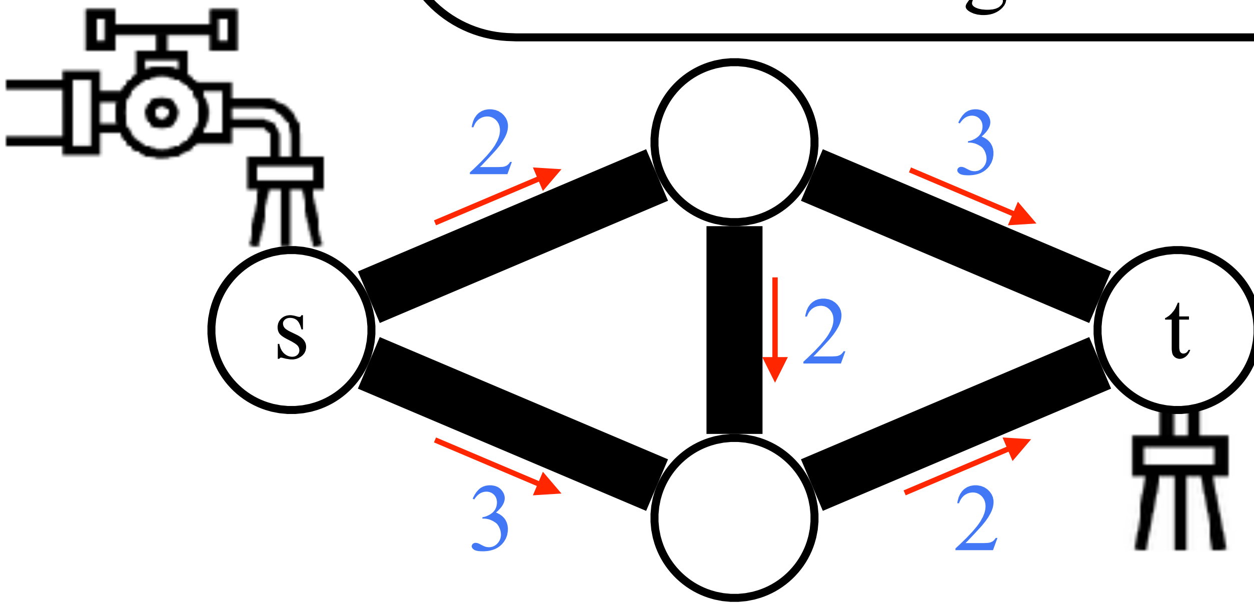
Problem

Input: a directed graph G and two distinguished nodes in G, which are a source node **s** and a sink node **t**. Every edge (u, v) in G has a nonnegative capacity c(u, v).

Output: the maximum flow from **s** to **t**.

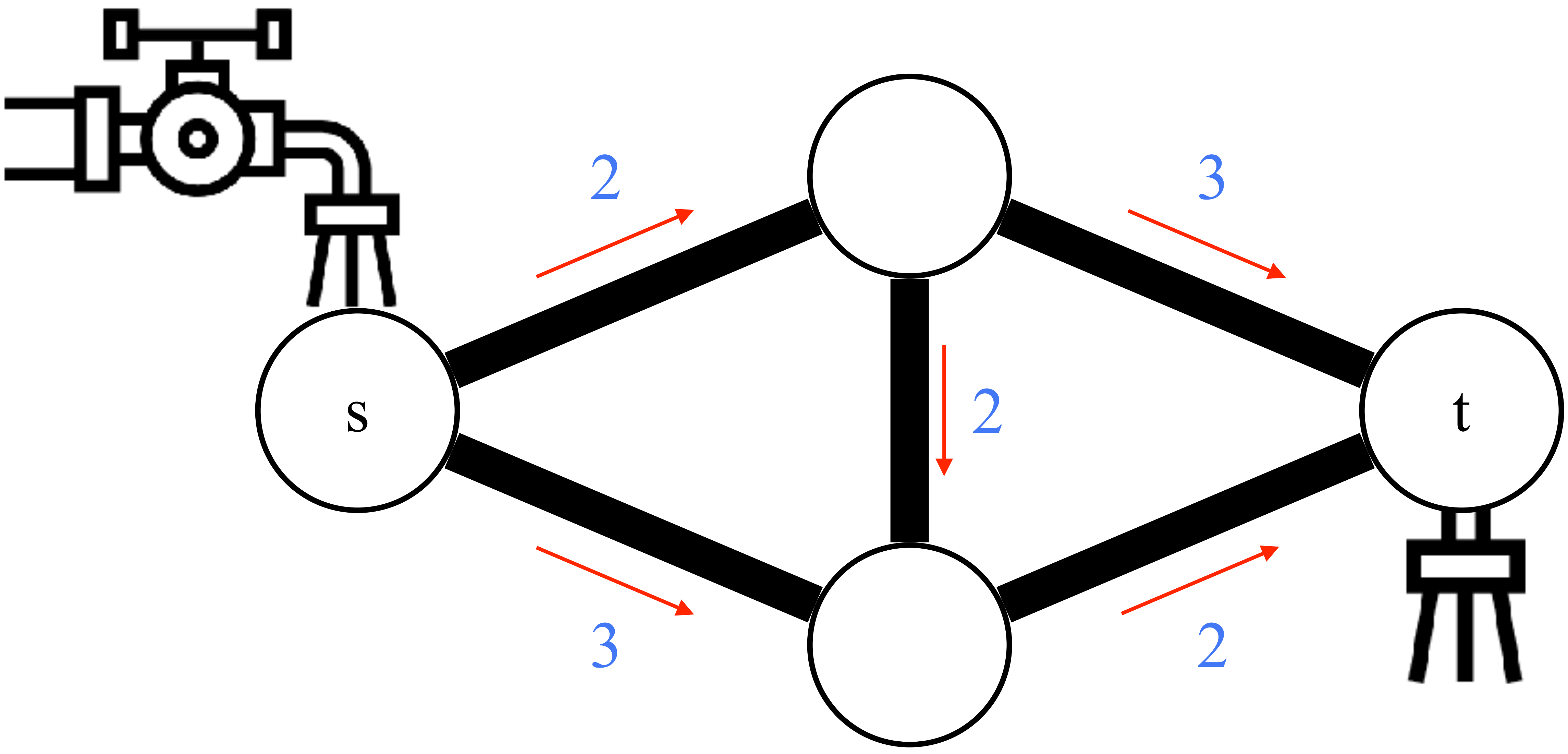
<u>Example</u>.

c(u, v) is the maximum rate that water can flow through the directed pipe (u, v).

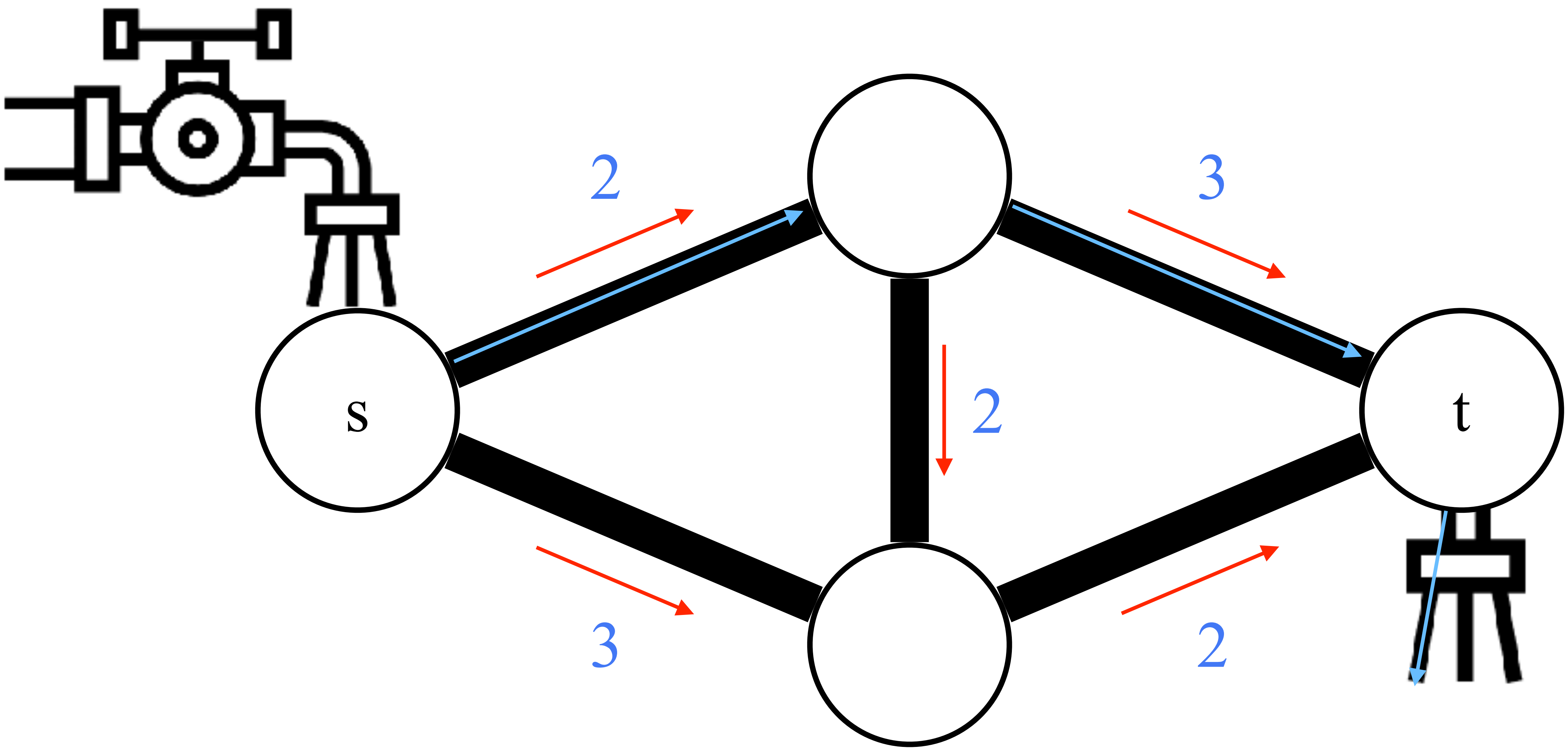


What is the maximum rate (gallon per second) that water can flow through the pipe network?

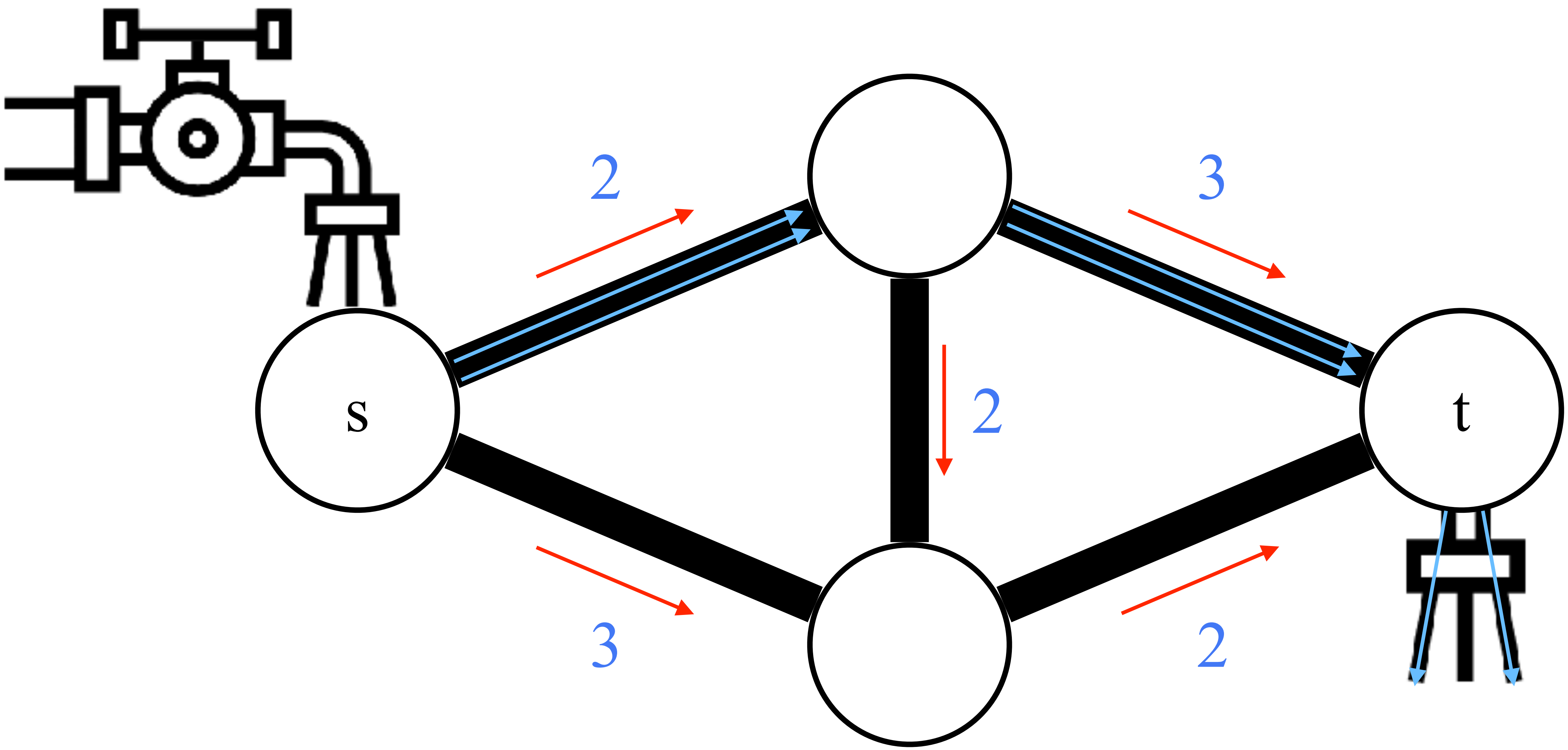
The first attempt



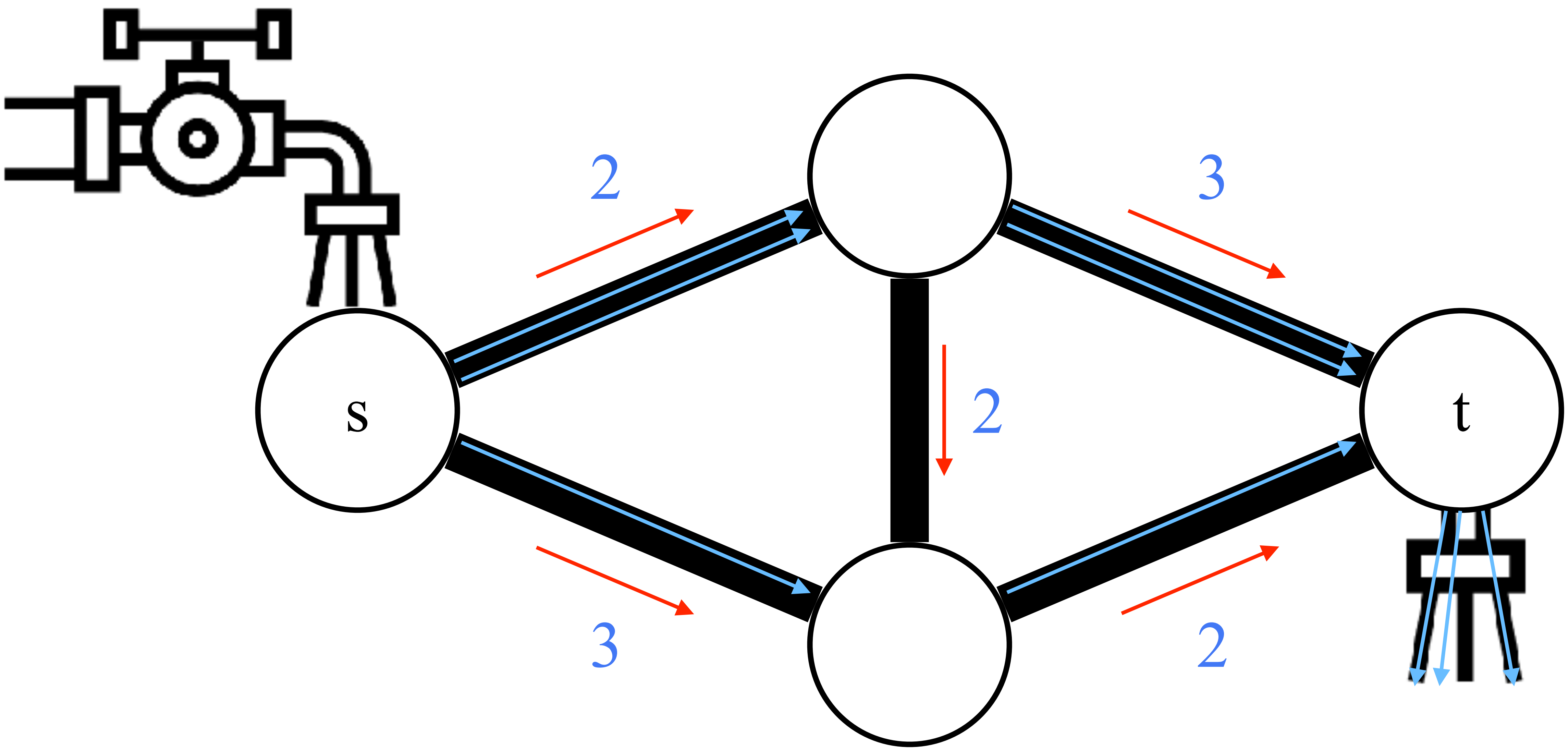
The first attempt



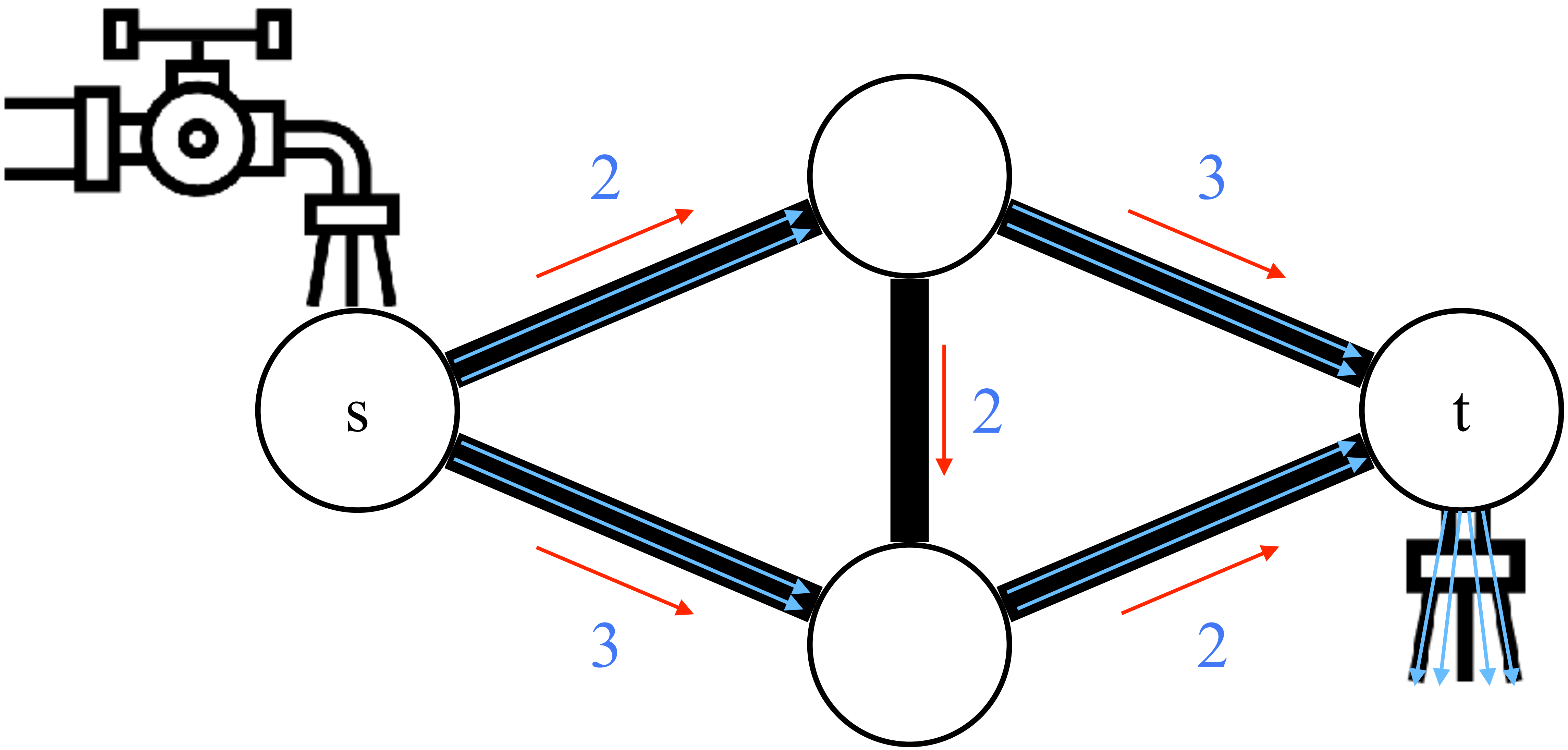
The first attempt



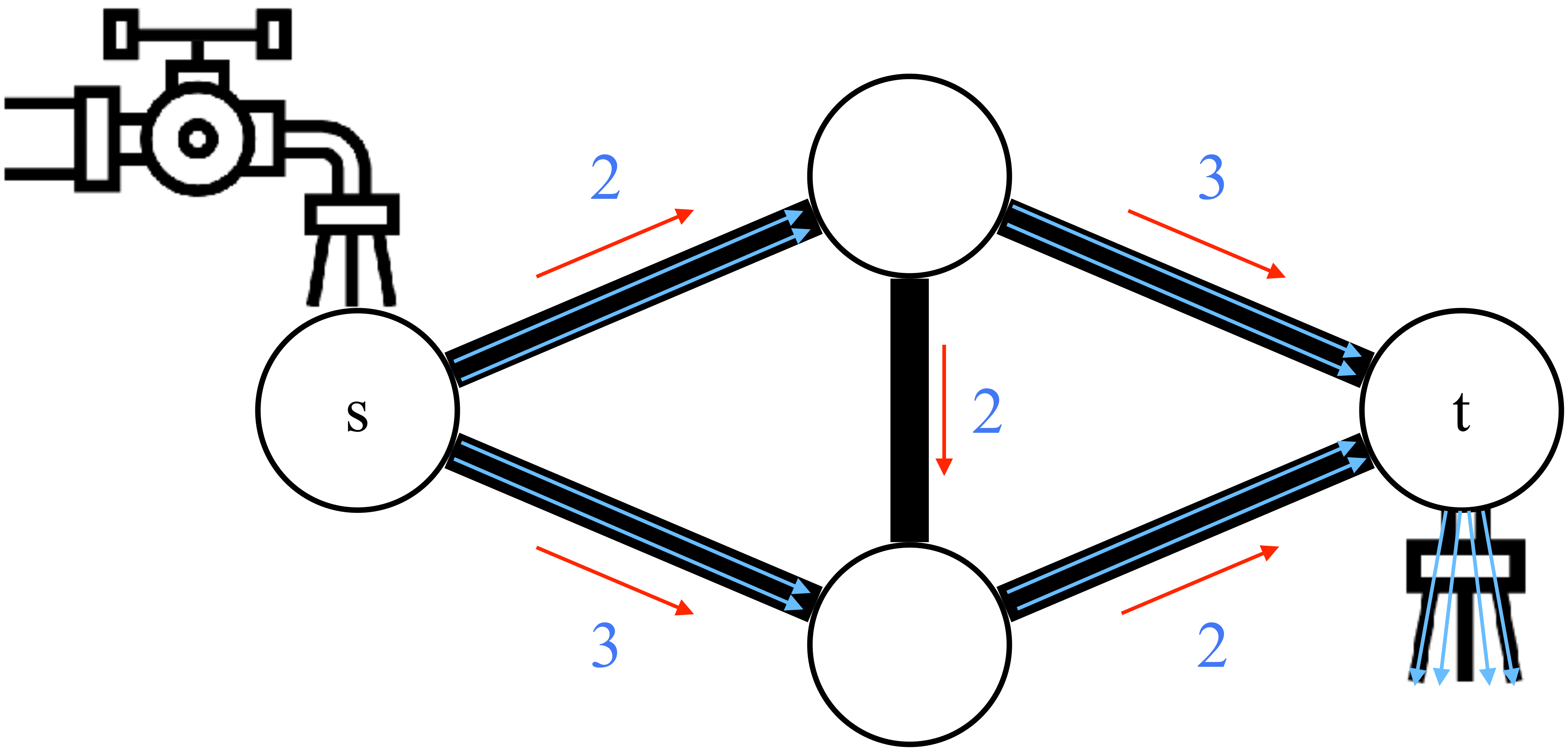
The first attempt



The first attempt

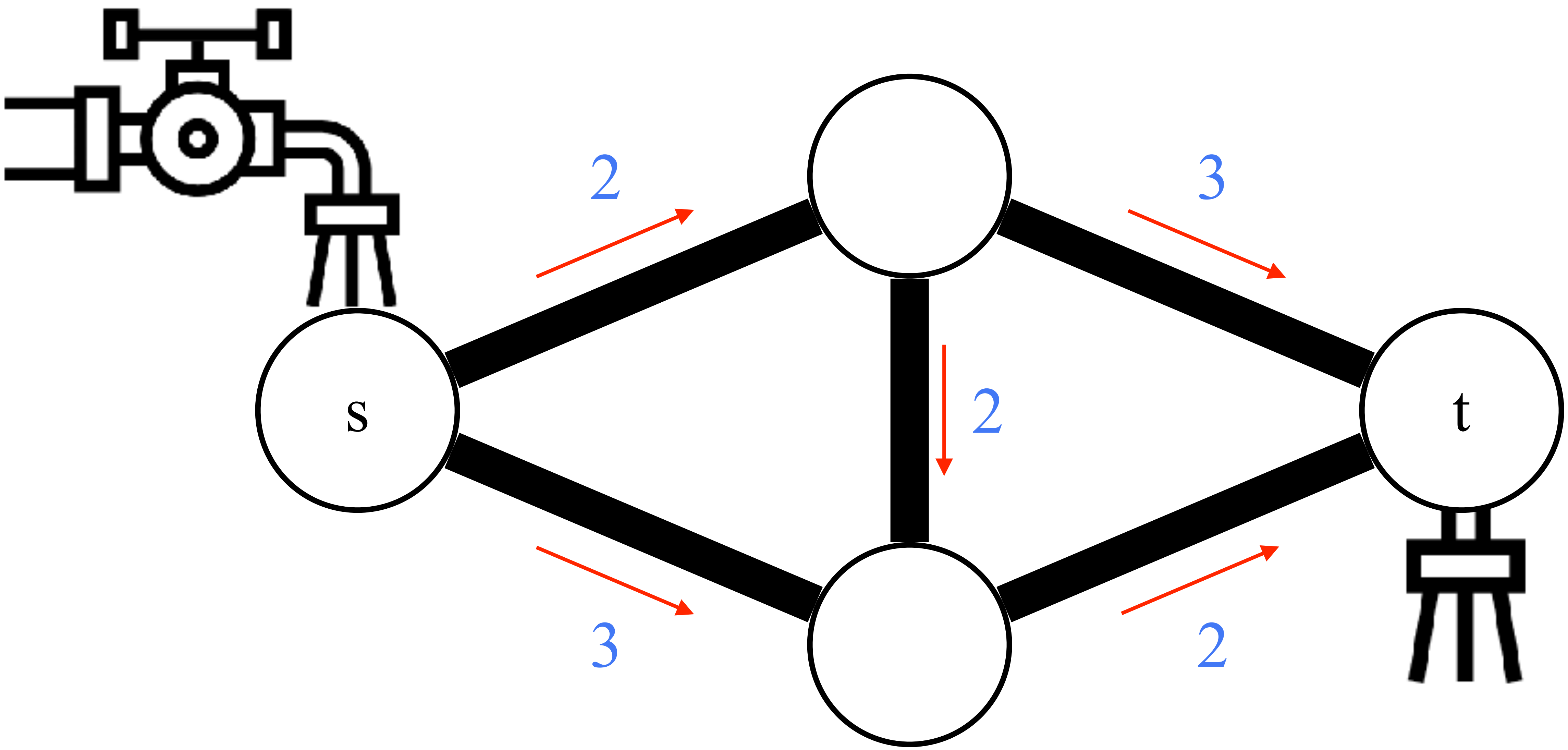


The first attempt

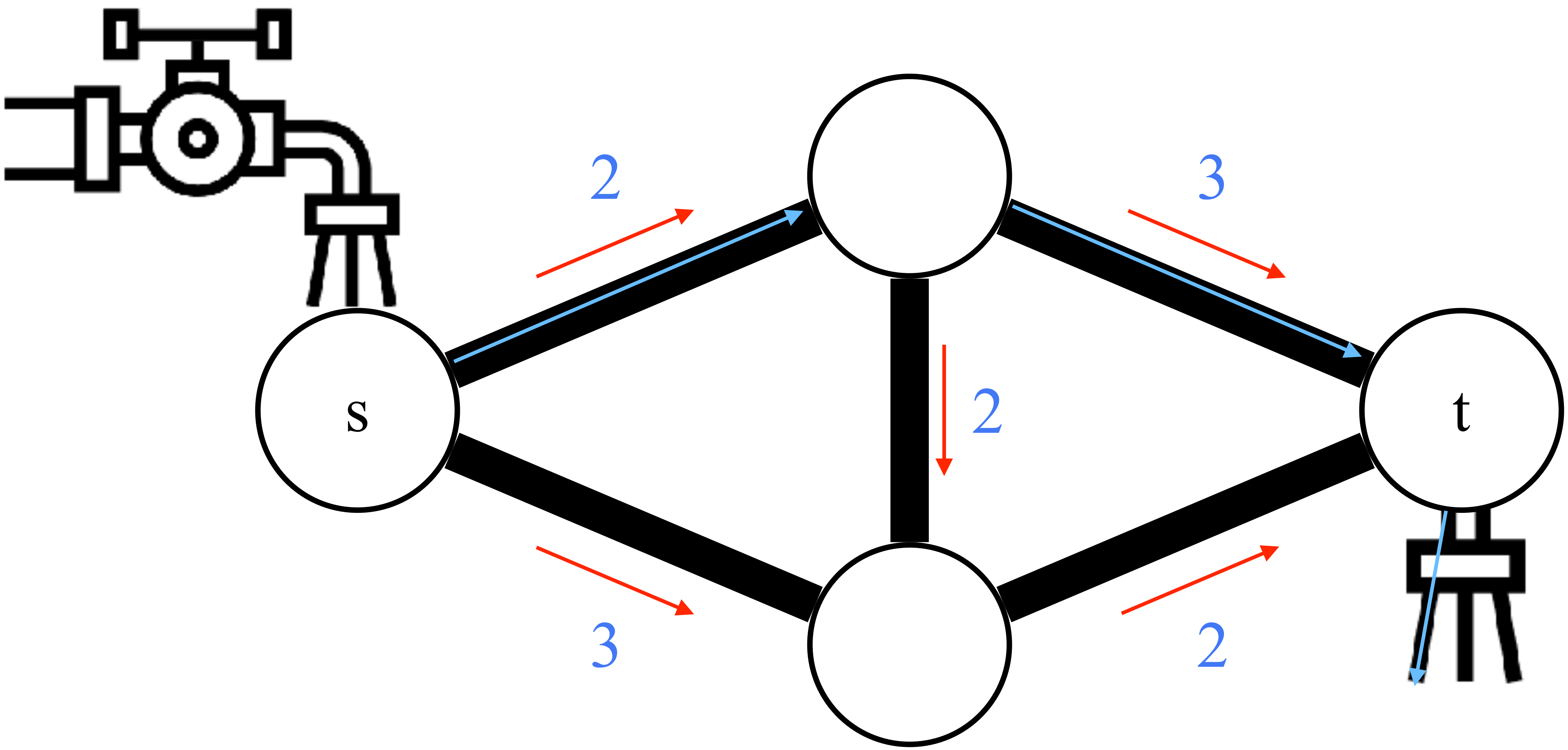


Iteratively pushing a 1-unit flow from s to t along some directed path seems yield the maximum flow.

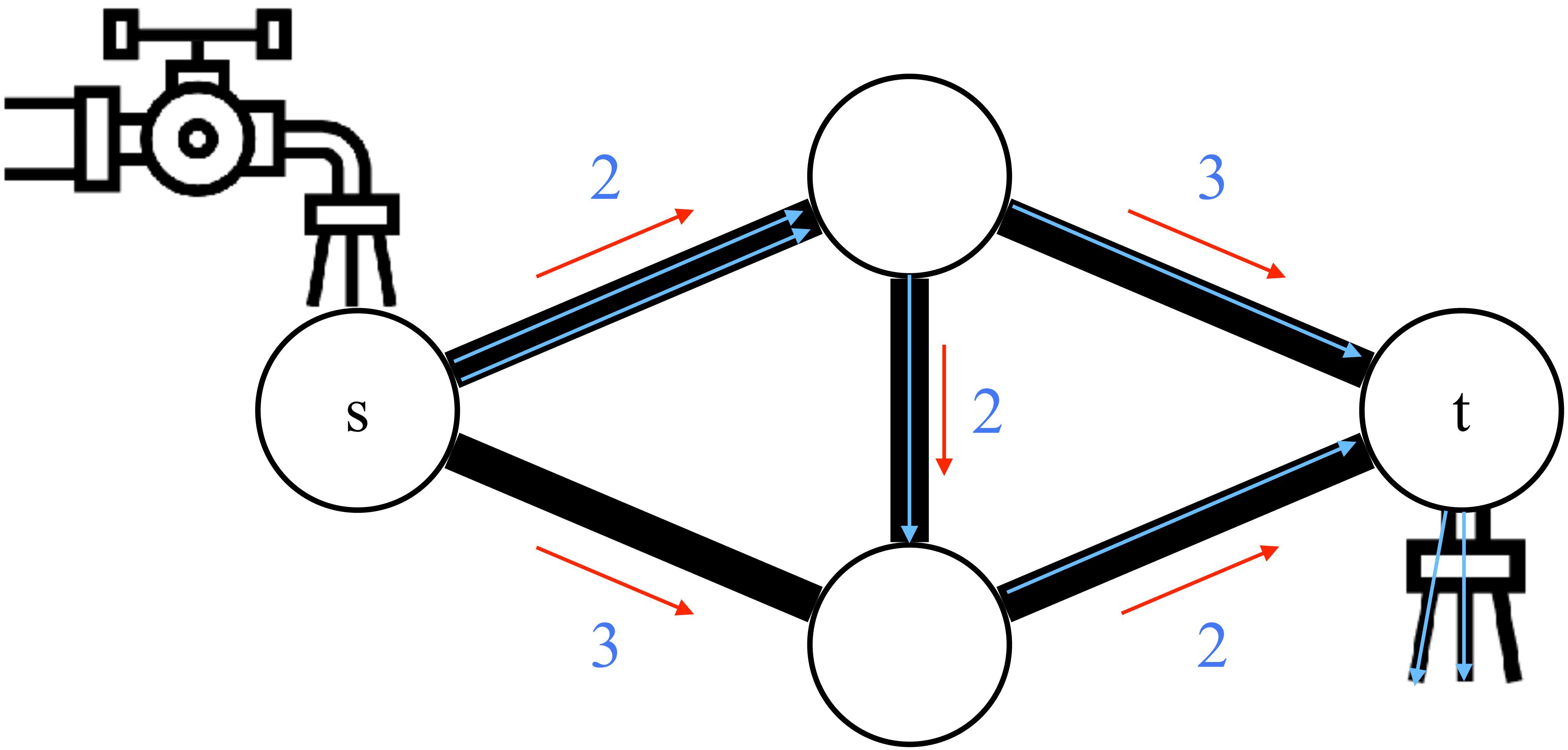
The first attempt may give a suboptimal solution



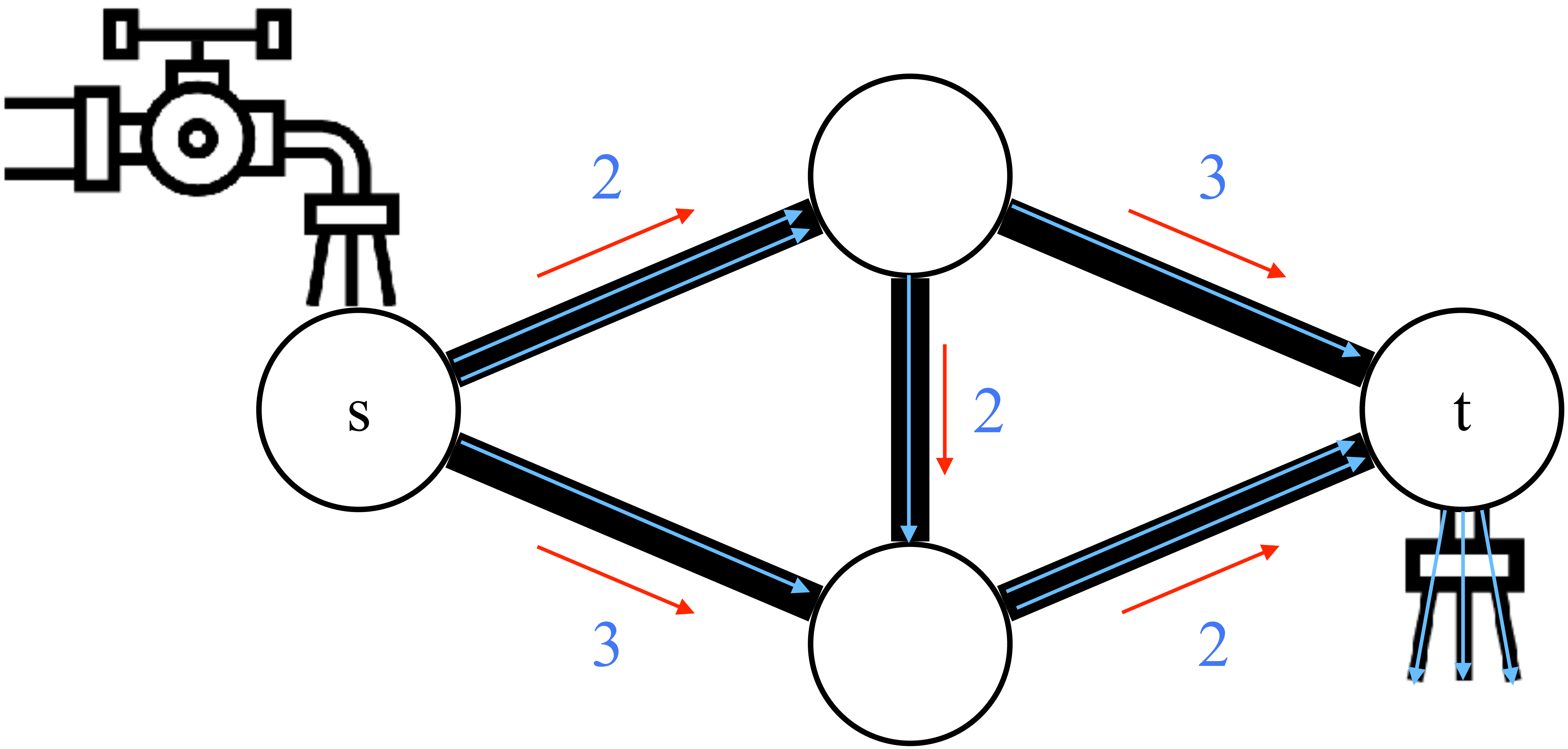
The first attempt may give a suboptimal solution



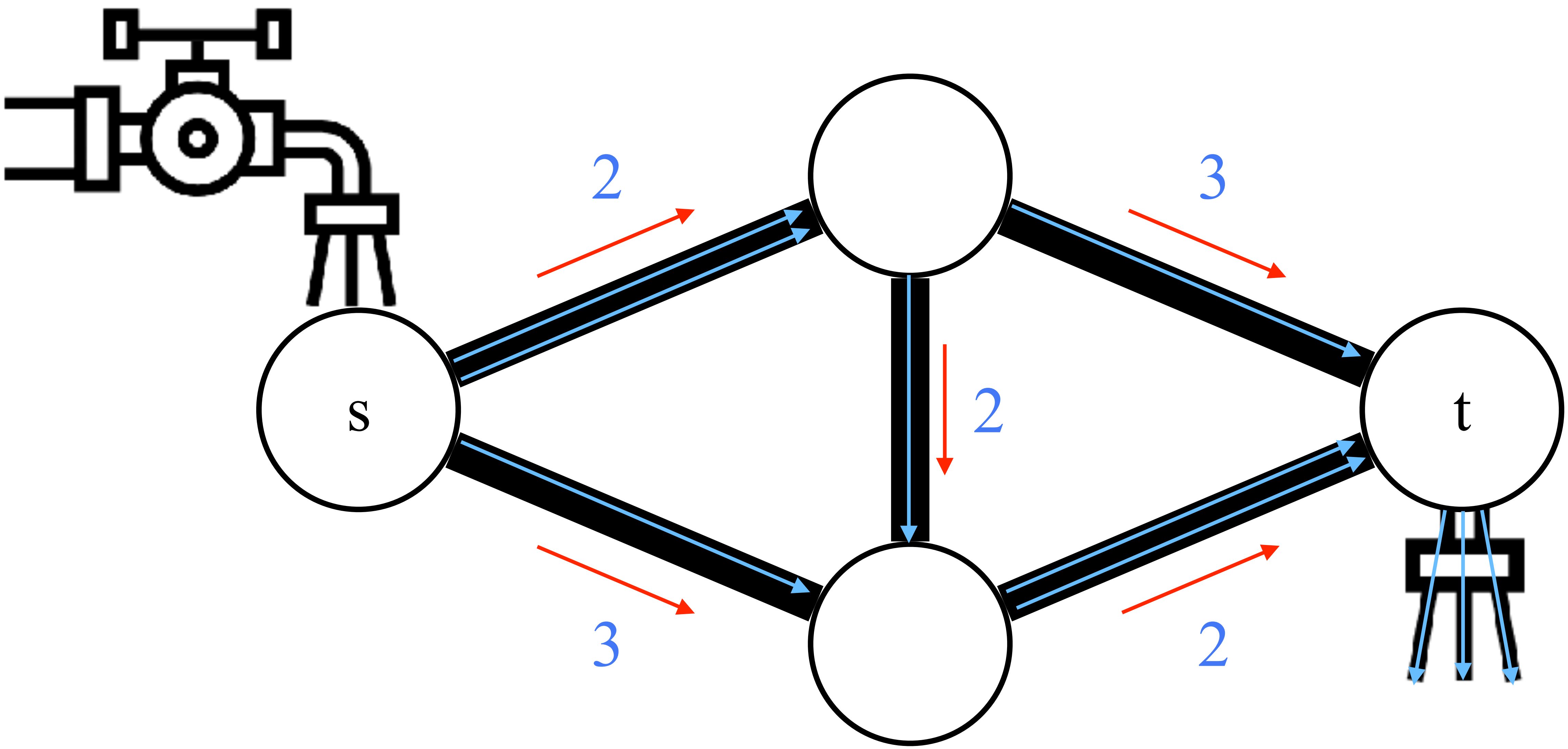
The first attempt may give a suboptimal solution



The first attempt may give a suboptimal solution

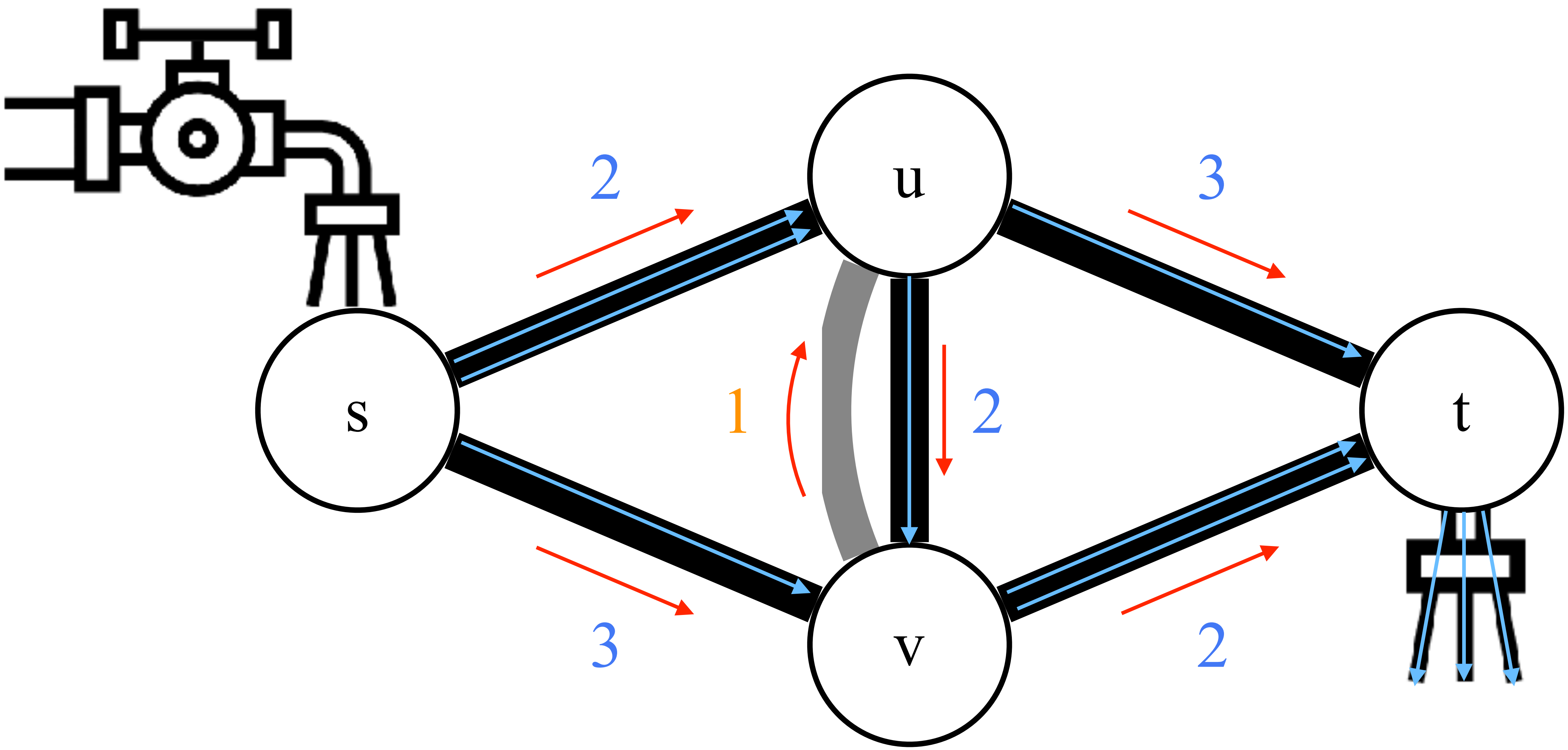


The first attempt may give a suboptimal solution



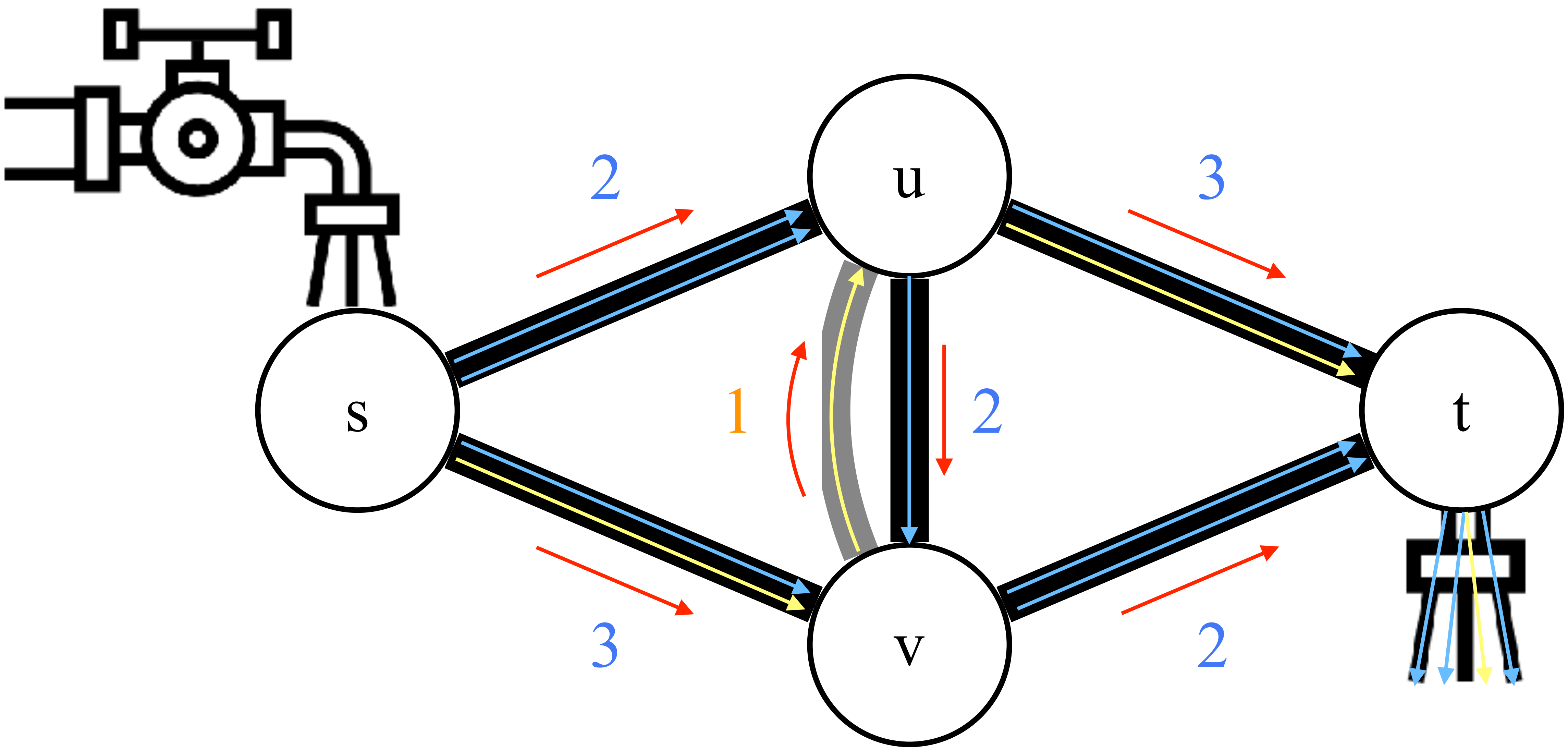
Iteratively pushing a 1-unit flow from s to t along some directed path may give a suboptimal flow.

The second attempt



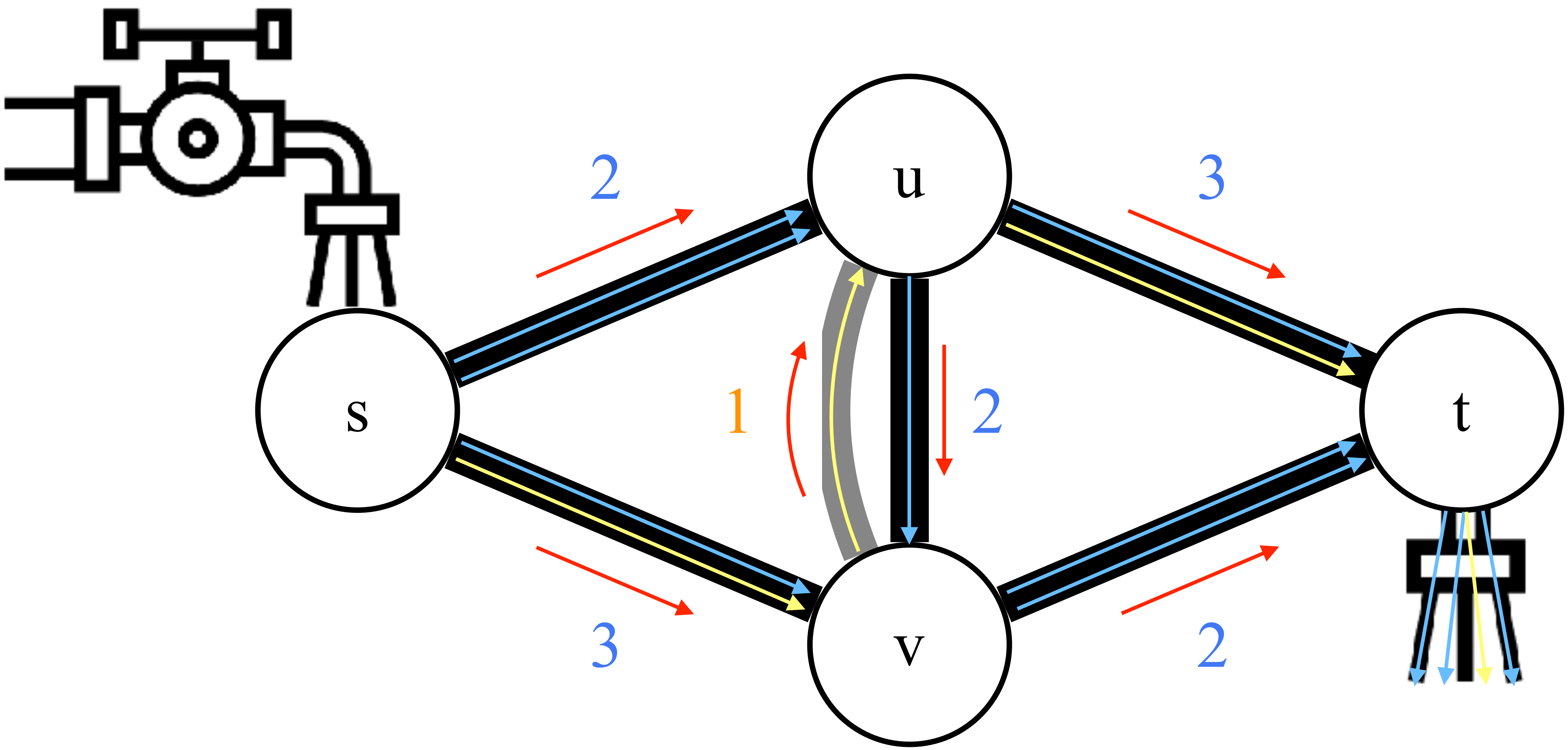
Build a reverse pipe (v, u) for each edge (u, v) that has k-unit water flowed through. Let c(v, u) = k.

The second attempt



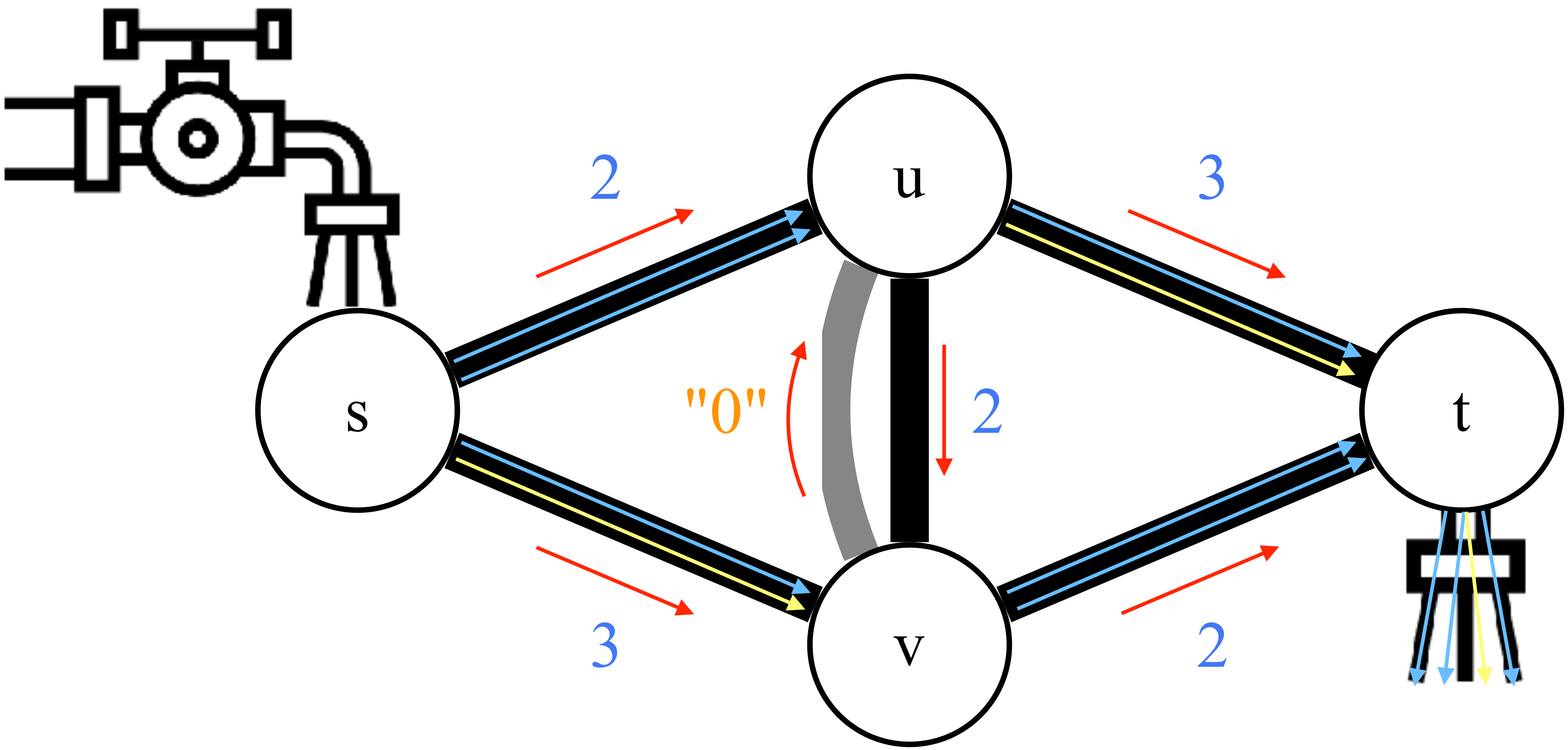
Build a reverse pipe (v, u) for each edge (u, v) that has k-unit water flowed through. Let c(v, u) = k.

The second attempt



If both u → v and v → u have some water flowed through, then the flows can cancel each other.

The second attempt



If both u → v and v → u have some water flowed through, then the flows can cancel each other.

The Ford-Fulkerson method

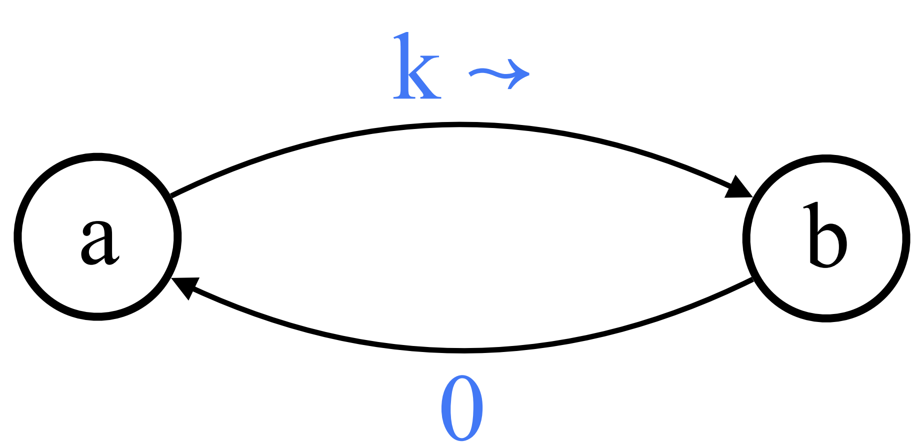
The second attempt is known as the Ford-Fulkerson method.

Notation f(a, b)

Since flow a → b can be cancelled with flow b → a, then at least one direction has no flow after the cancellation. To simplify the latter explanation, by

$$f(a, b) = -f(b, a) = k \geq 0$$

we denote that there is a k-unit flow a → b and there is no flow b → a.

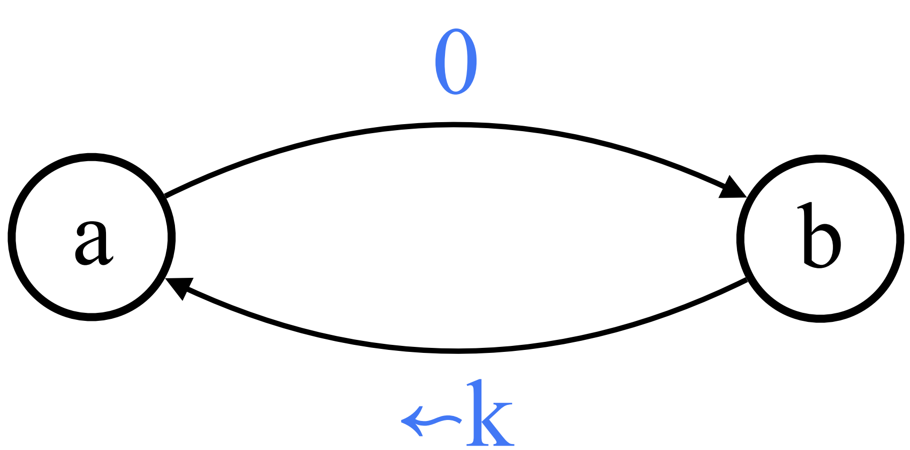


Notation f(a, b)

Since flow a → b can be cancelled with flow b → a, then at least one direction has no flow after the cancellation. To simplify the latter explanation, by

$$f(b, a) = -f(a, b) = k \geq 0$$

we denote that there is a k-unit flow b → a and there is no flow a → b.

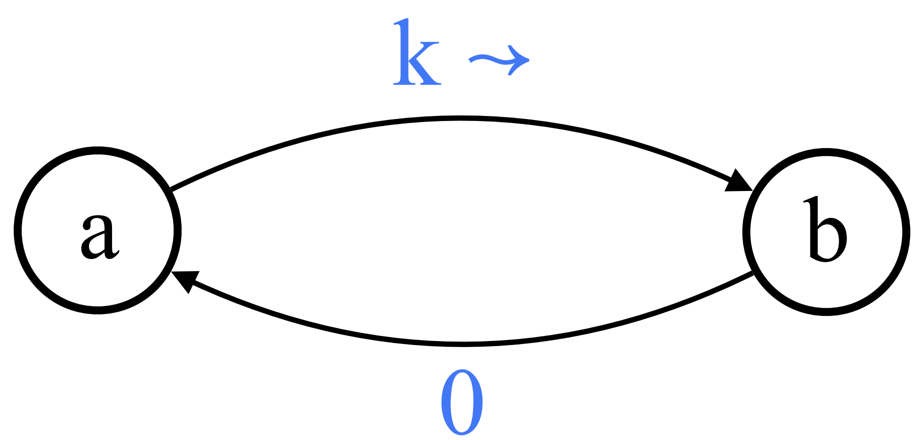


Notation c(a, b)

The directed pipes (a, b) and (b, a) may have different capacity. Hence, c(a, b) may be not equal to c(b, a).

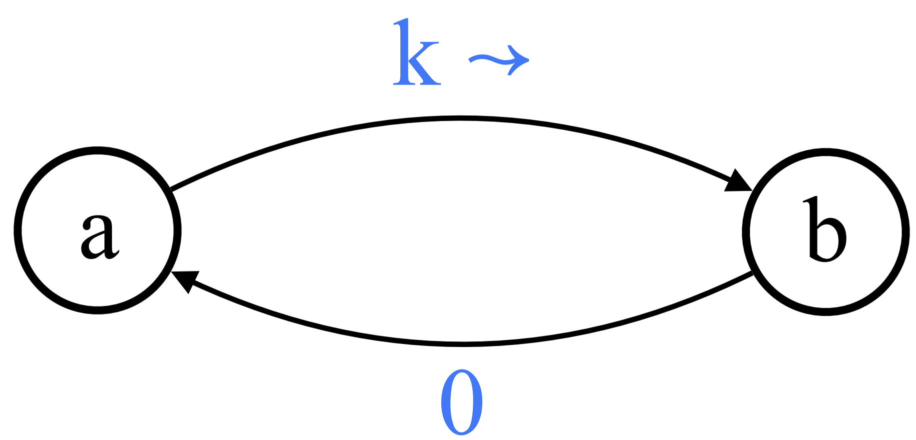
If edge $(a, b) \notin G$, then $c(a, b) = 0$.

Notation c(a, b)



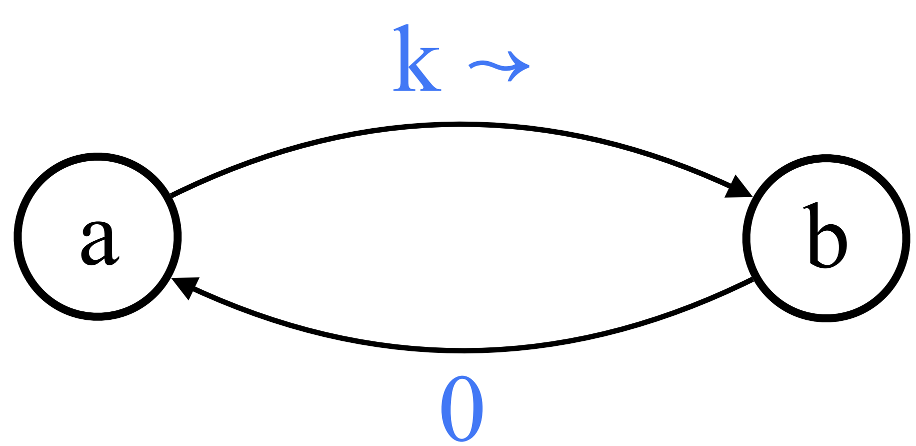
Suppose we have k-unit water flowed through the directed edge (a, b), then $k \leq c(a, b)$.

Notation c(a, b)



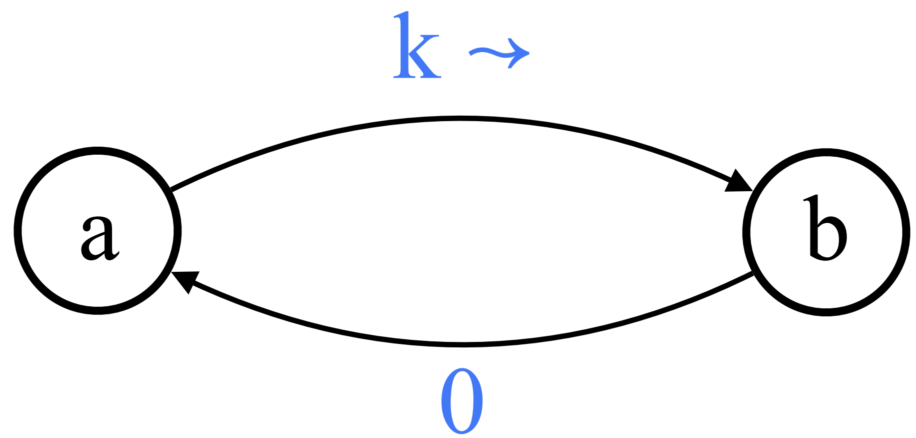
The flow on edge (a, b) can be further increased by
$c(a, b) - k \geq 0$ unit.

Notation c(a, b)



The flow on edge (b, a) can be further increased by
$c(b, a)+k \geq 0$ unit.

Notation $c_f(a, b)$



We could say, in other words, that:

[1] The flow on edge (a, b) can be increased by at most
$c_f(a, b) \equiv c(a, b) - f(a, b) \geq 0$ units.

[2] The flow on edge (v, u) can be increased by at most
$c_f(b, a) \equiv c(b, a) - f(b, a) \geq 0$ units.

The residual network G_f

$G_f = G$ except that:

each edge (a, b) in G has a capacity $c(a, b)$, but

each edge (a, b) in G_f has a capacity $c_f(a, b) = c(a, b) - f(a, b)$.

The Ford-Fulkerson method

```
Ford-Fulkerson(G, s, t){
    foreach edge (a, b) in G{
        f(a, b) ← 0;
        f(b, a) ← 0;
    }
    while ∃ an augmenting (simple) path P from s to t in G_f{
        c_f(P) ← min{c_f(a, b) : (a, b) in P}; // has c_f(P) > 0
        foreach edge (a, b) in P{
            f(a, b) ← f(a, b) + c_f(P);  // increase a c_f(P)-unit flow
            f(b, a) ← - f(a, b);            along the path P
        }
    }
}
```

Finding an augmenting path P from s to t

	By DFS-Visit(s)	By BFS(s)
running time	$O((m+n)\|f^*\|)$	$O((m+n)nm)$

aka Edmonds-Karp algorithm

$|f^*|$ denotes the quantity of the maximum flow f^*.

The Correctness of the Ford-Fulkerson Method

Properties

[1] Capacity constraint:

for every edge (a, b), $f(a, b) \le c(a, b)$.

[2] Skew symmetry:

for every edge (a, b), $f(a, b) = - f(b, a)$.

[3] Flow conservation:

for every node a in V - {s, t}, $f(V, a) \equiv \sum_v f(v, a) = 0$.

// We **cannot accumulate** some water at a node other than s and t. As time increases, the amount of water accumulated at the node goes to infinity.

[4] The amount of water flowed in equals the amount of water flowed out, $f(s, V) \equiv \sum_v f(s, v) = |f| = f(V, t) \equiv \sum_v f(v, t)$.

Cut

Let S and T be a partition of V(G); that is, $S \cup T = V(G)$ and $S \cap T = \varnothing$.

Given a flow f, let S be the set of nodes that are reachable from s via a sequence of directed edges (a, b) whose $c_f(a, b) > 0$. Let T be $V \setminus S$.

Let further $C(S, T) \equiv \sum_{a \in S} \sum_{b \in T} c(a, b)$.

Max-flow min-cut theorem

[1] f is a maximum flow in G.
[2] G_f contains no augmenting path from s to t.
[3] $|f| = c(S, T)$ for some cut (S, T) of G.

Claim. [1] $\Leftrightarrow$ [2] $\Leftrightarrow$ [3].

[2] $\Rightarrow$ [1] proves the correctness of the Ford-Fulkerson method. We plan to prove that [1] $\Rightarrow$ [2] $\Rightarrow$ [3] $\Rightarrow$ [1].

Max-flow min-cut theorem

[1] f is a maximum flow in G.
[2] G_f contains no augmenting (simple) path from s to t.
[3] $|f| = c(S, T)$ for some cut (S, T) of G.

Proof of [1] ⇒ [2].

If f is a maximum flow, but G_f has some augmenting path P from s to t. The existence of P implies that the existence of a augmenting (simple) path P' from s to t. If we augment P' to the flow f, then the resulting flow f' has the quantity

$$|f'| = \sum_{v \in V} f'(s, v) = |f| + c_f(P')$$

because P' leaves s at the beginning and never comes back. Since $c_f(P') > 0$, $|f'| > |f|$ $\rightarrow\leftarrow$.

Max-flow min-cut theorem

[1] f is a maximum flow in G.
[2] G_f contains no augmenting path from s to t.
[3] $|f| = c(S, T)$ for some cut (S, T) of G.

Proof of [2] $\Rightarrow$ [3].

If there exist some node a in S and some node b in T that

$$f(a, b) < c(a, b),$$

then b is in S $\rightarrow\leftarrow$. Thus, $f(a, b) = c(a, b)$ for every a in S, b in T.

Since there exists no augmenting path in G_f from s to t, $T \neq \varnothing$. Hence, $f(s, V) = f(S, T) = c(S, T)$.

Max-flow min-cut theorem

[1] f is a maximum flow in G.
[2] G_f contains no augmenting path from s to t.
[3] $|f| = c(S, T)$ for some cut (S, T) of G.

Proof of [3] ⇒ [1].

Given f and S, T w.r.t f, we have for any flow g

$|g| = g(S, T) = \sum_{a \in S} \sum_{b \in T} g(a, b) \leq \sum_{a \in S} \sum_{b \in T} c(a, b)$.

By [3], $|f| = \sum_{a \in S} \sum_{b \in T} c(a, b)$, we have $|g| \leq |f|$.

Thus, f is a maximum flow.

Exercise*

Use the Ford-Fulkerson method to solve the Programming Assignment 3-C (Card Game).

In the Programming Quiz, we will explain how to reduce the problem into a flow problem. You may need to use Ford-Fulkerson algorithm to solve it.

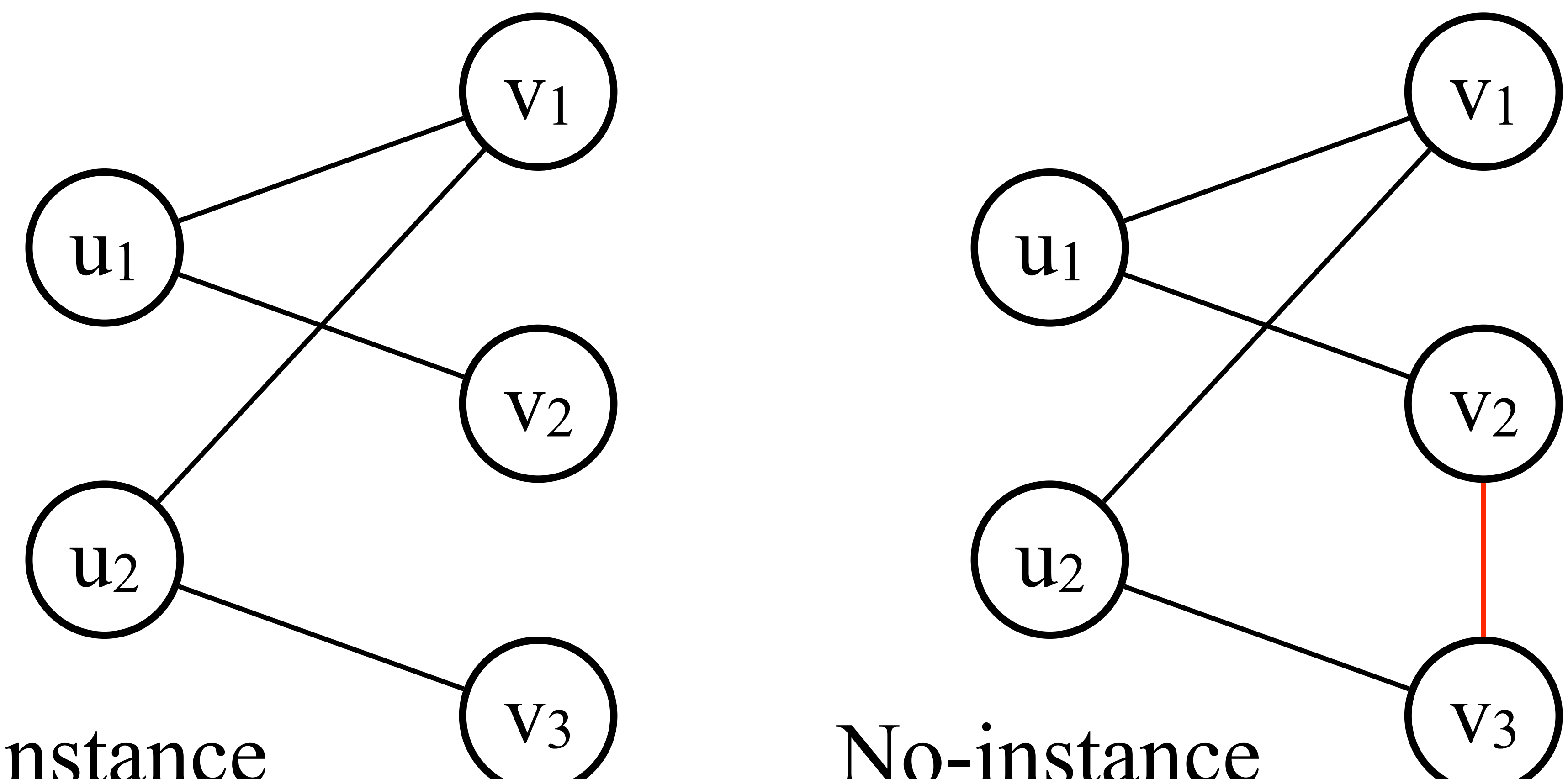
Some Applications of Maximum Flows

Bipartite Matching

Bipartite Graphs

A graph G is bipartite if its node set can be partitioned into two subsets U, V so that every edge in G has exactly one end-point in U.

Example.

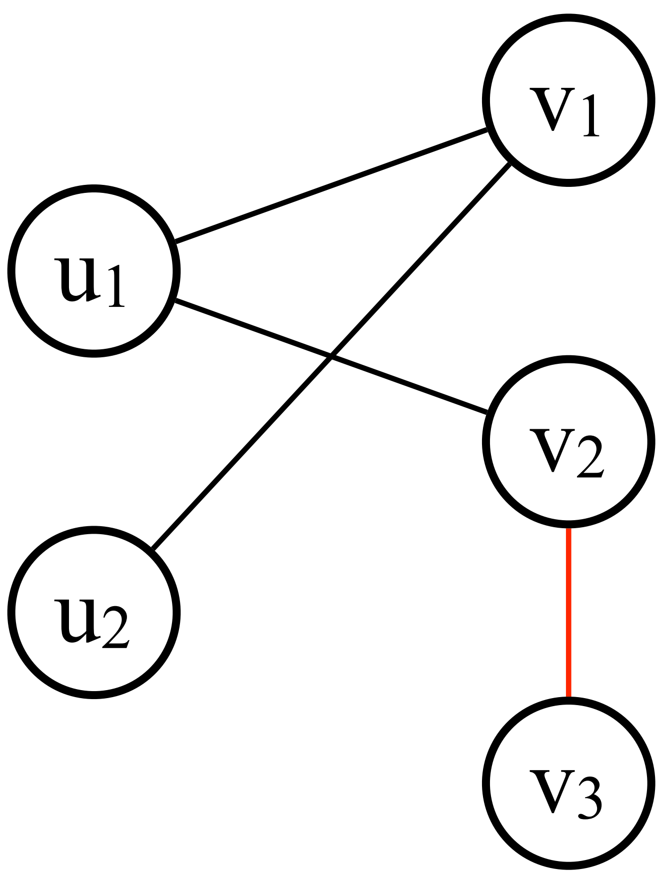


Yes-instance No-instance

Bipartite Graphs

A graph G is bipartite if its node set can be partitioned into two subsets U, V so that every edge in G has exactly one end-point in U.

Example.



Is this graph bipartite?

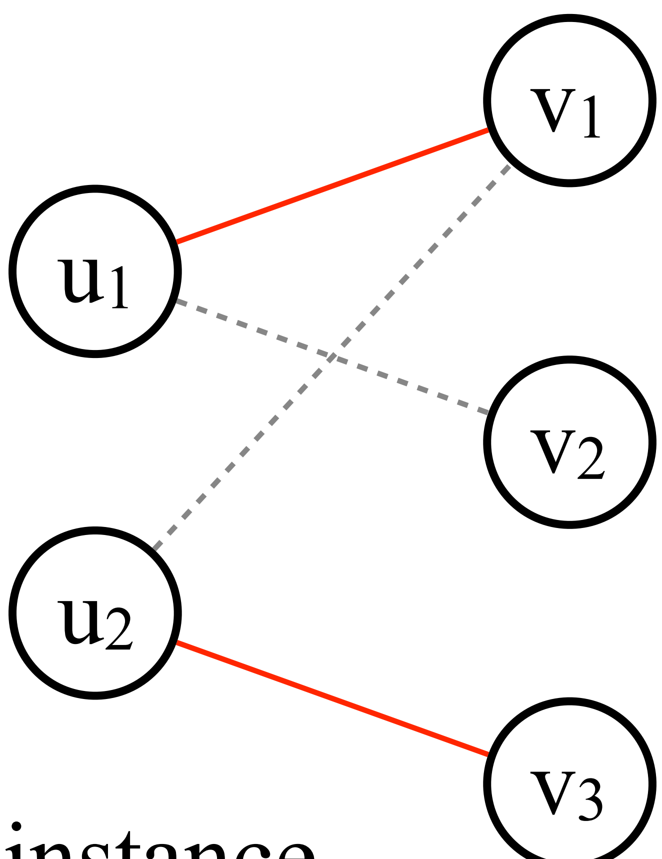
Exercise

Give an algorithm to test whether the input graph G is bipartite or not.

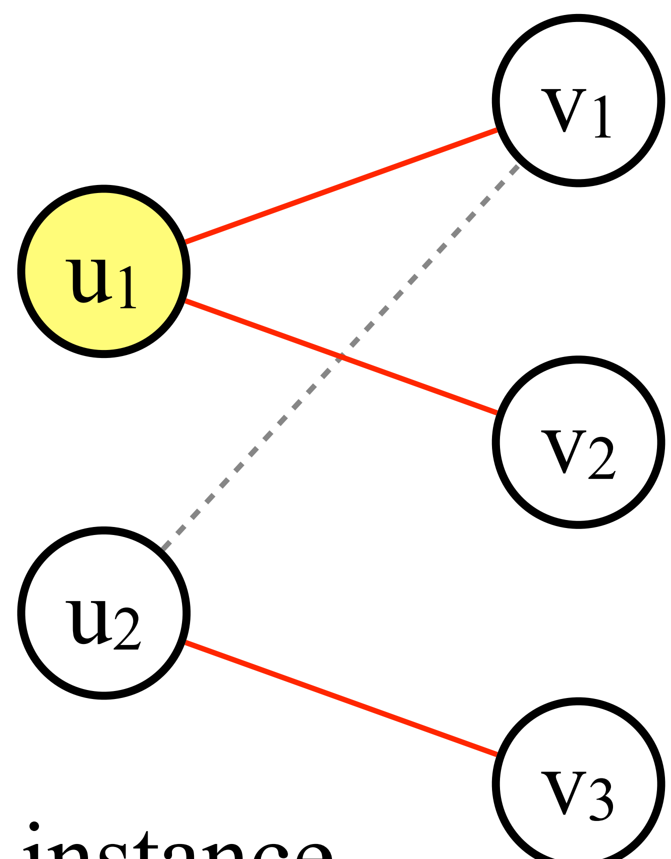
Matching

A matching M is a set of edges so that every two edges in M share no endpoints.

Example.



Yes-instance



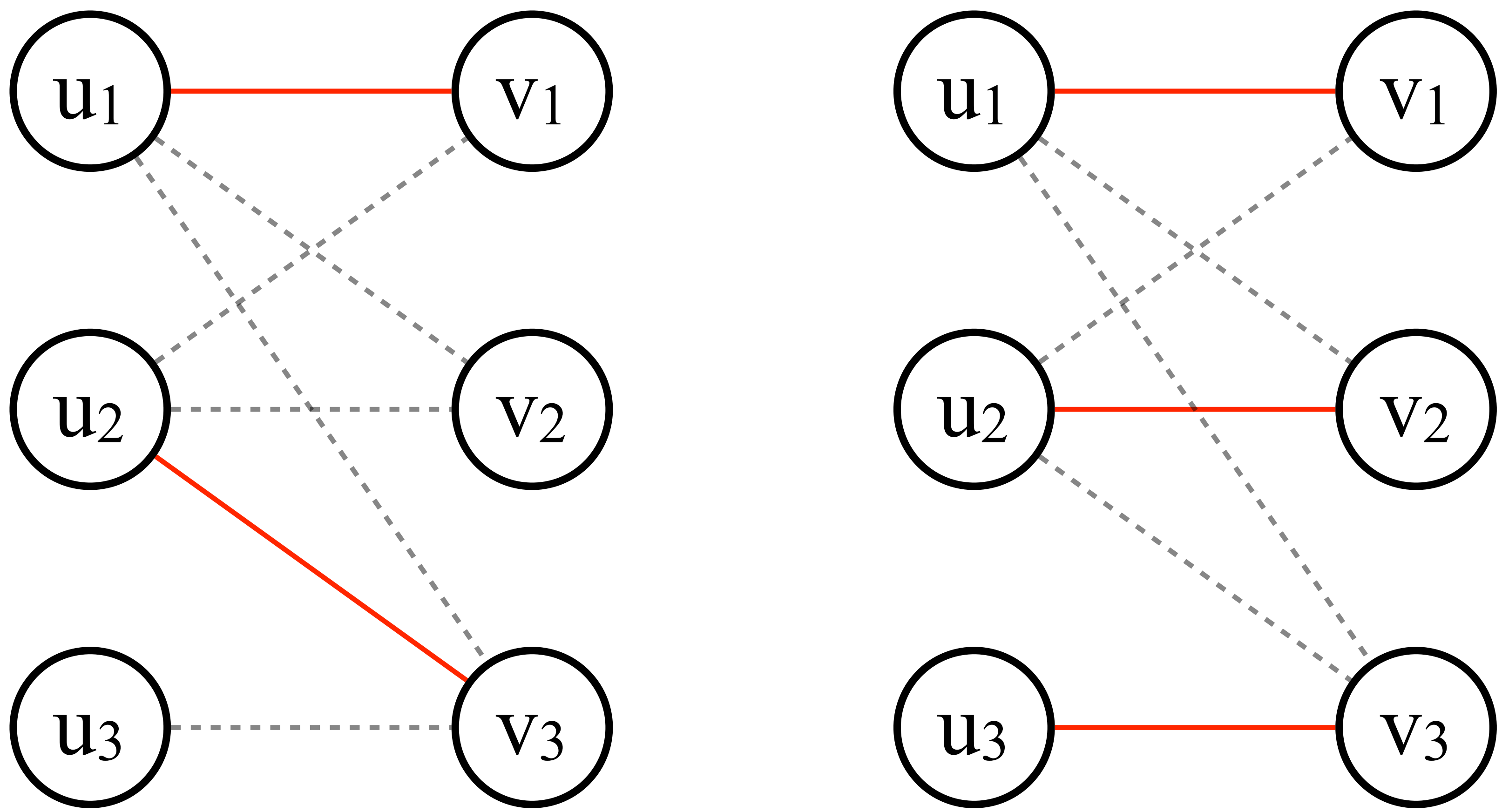
No-instance

Bipartite Matching

Input: an undirected bipartite graph $G = (U \cup V, E)$.

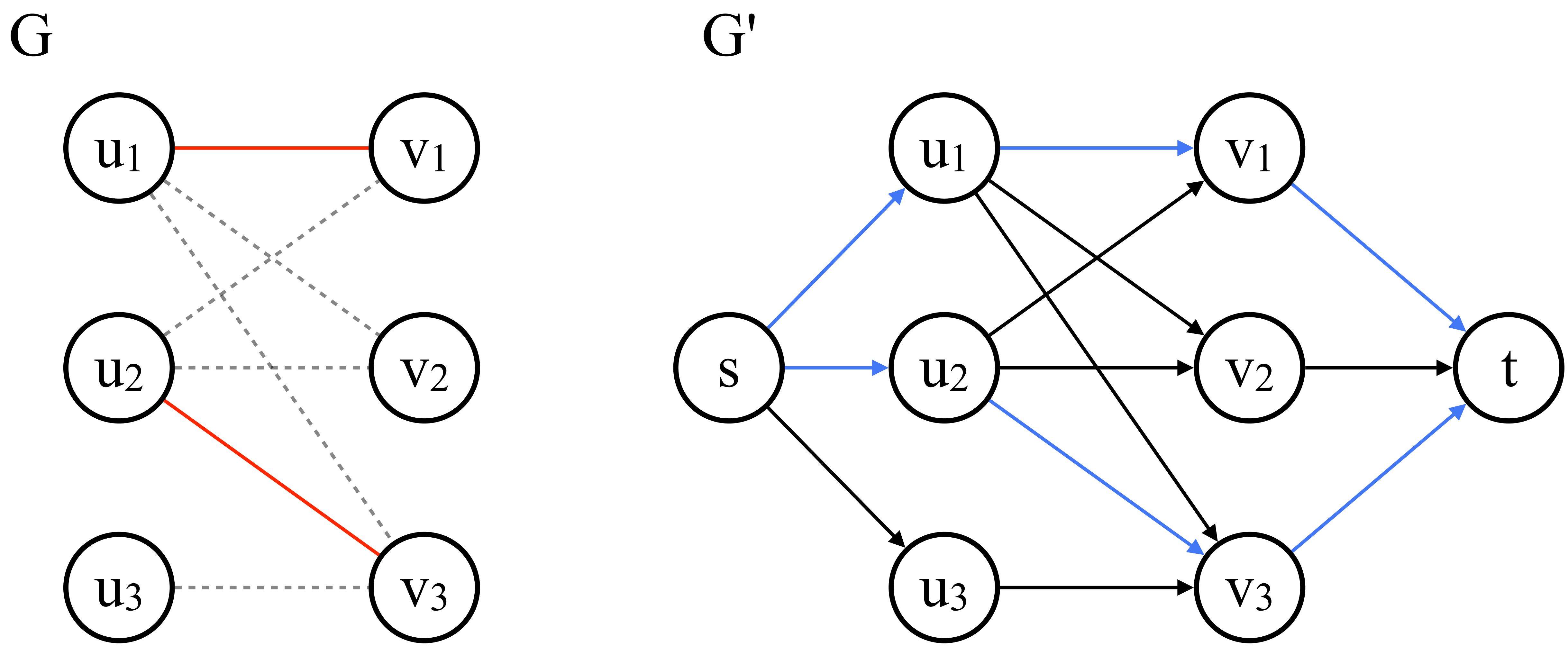
Output: a matching M so that $|M|$ is maximized.

Example.



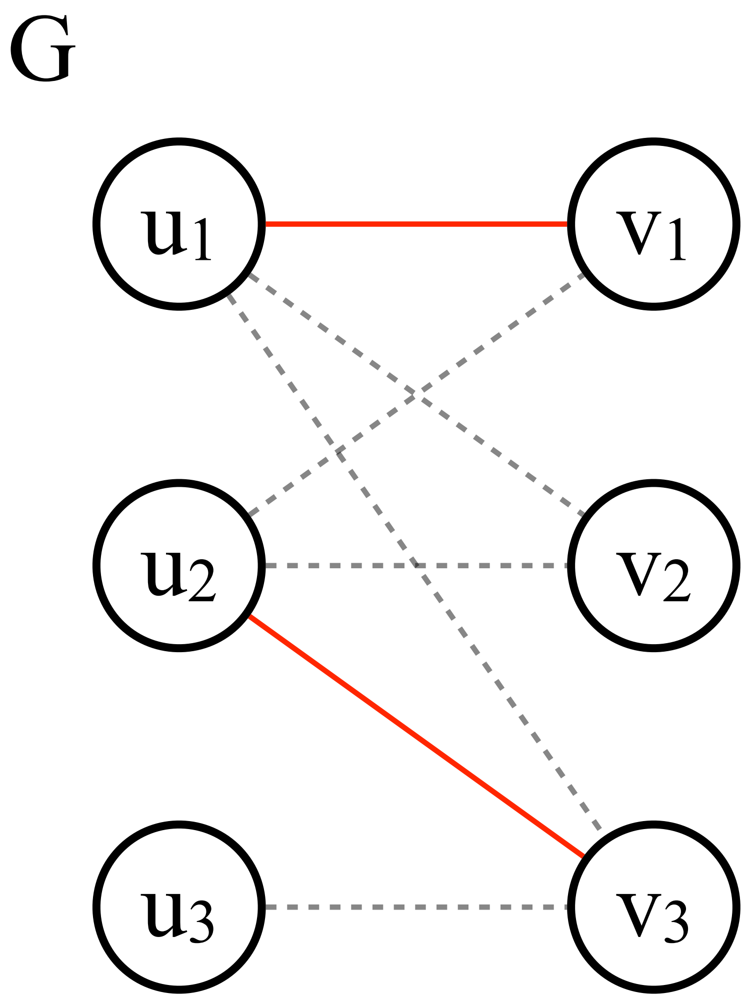
$|M| = 2$ (suboptimal) $|M| = 3$ (optimal)

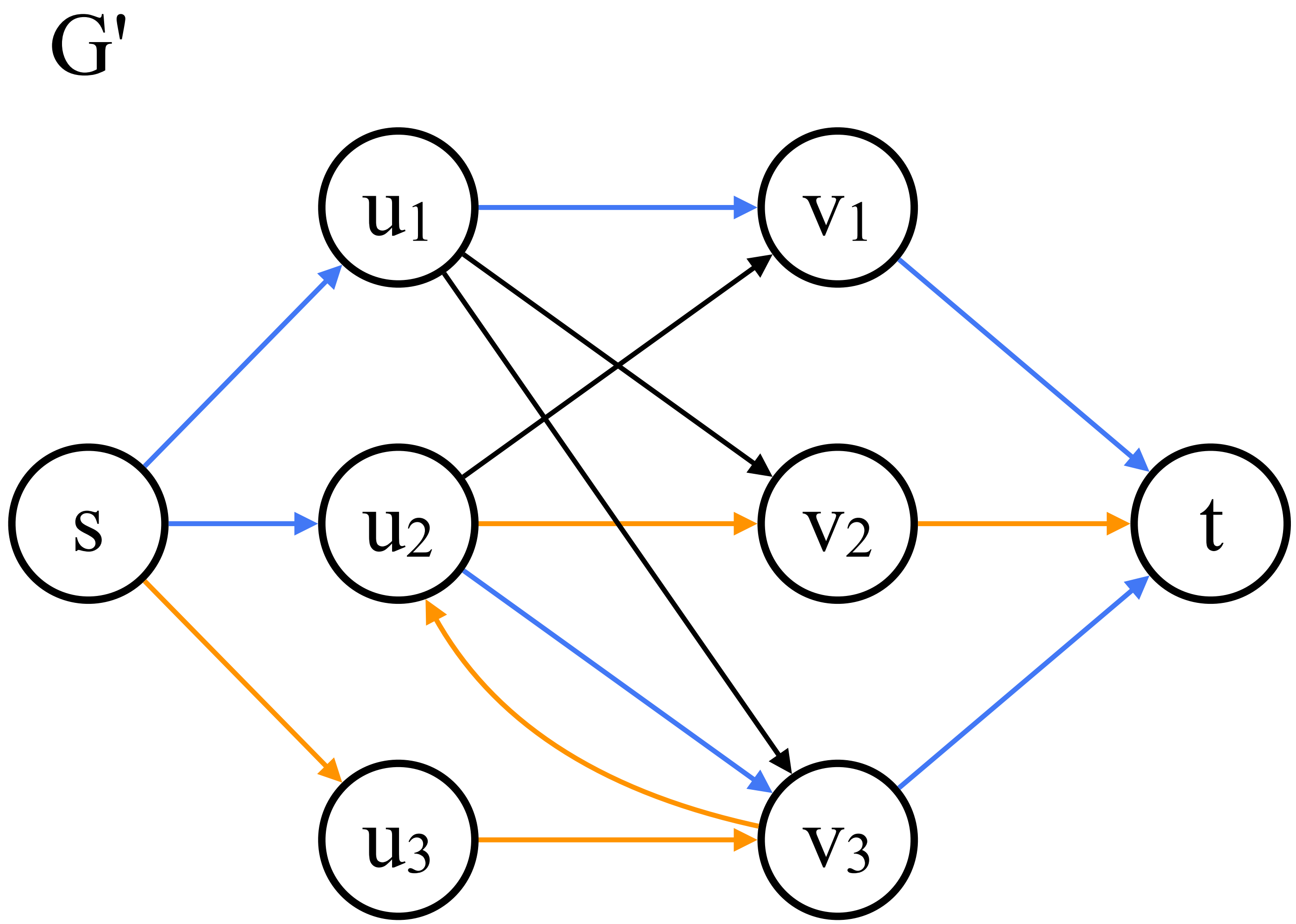
Reduce Bipartite Matching to Maximum Flow



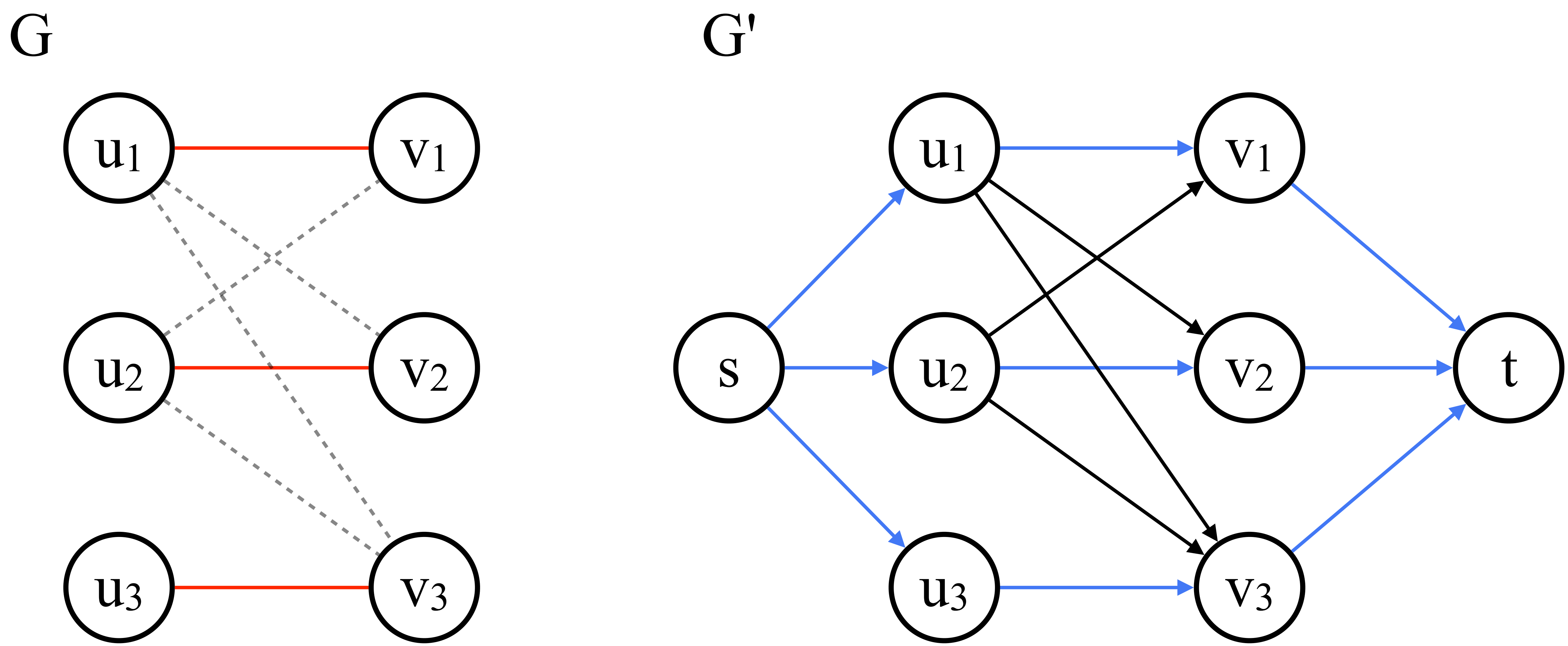
Let each directed edge (u, v) in G'
has capacity c(u, v) = 1.

Reduce Bipartite Matching to Maximum Flow





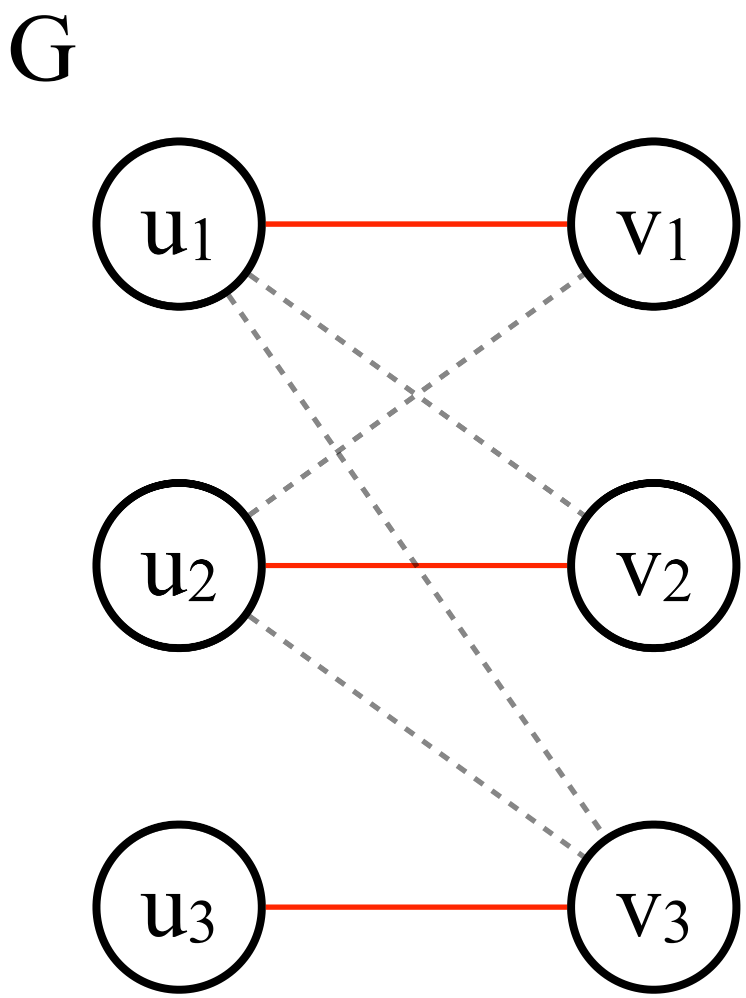
Reduce Bipartite Matching to Maximum Flow

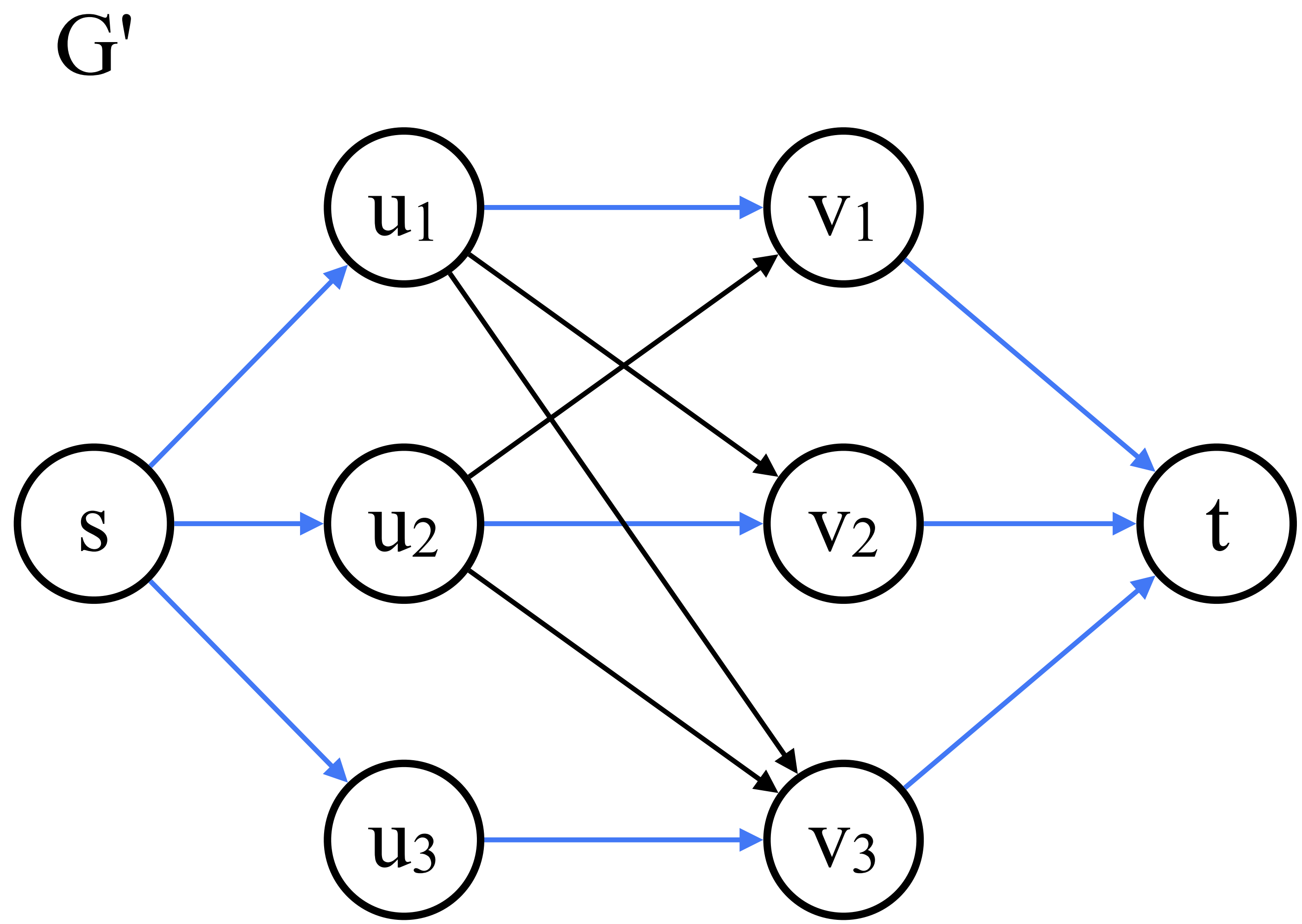


G has matching M of size k iff G' has a flow of k units.

$\Rightarrow$ clearly holds.

Reduce Bipartite Matching to Maximum Flow

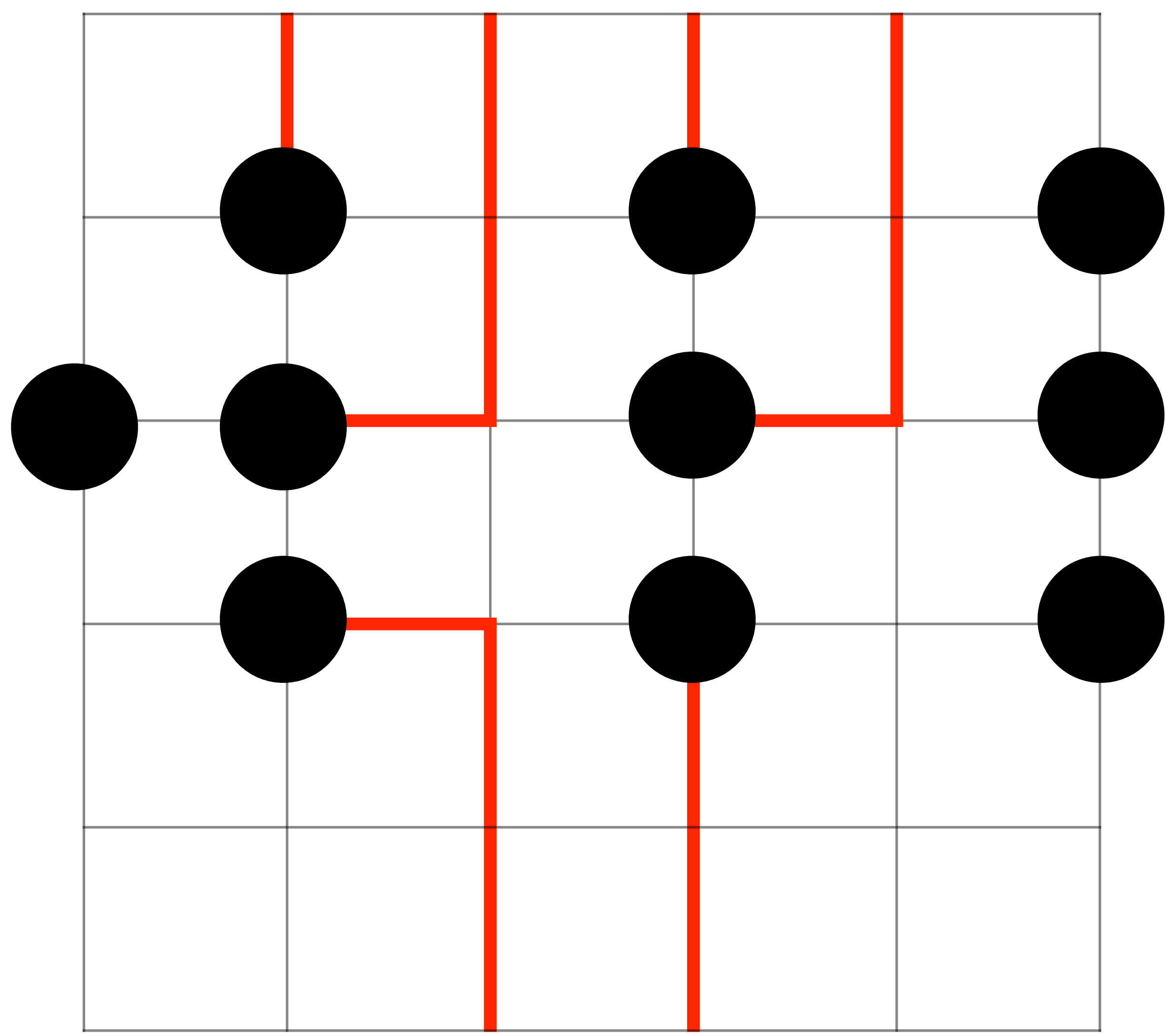




$\Leftarrow$ holds because every unit flow in G' is a directed path $s \to u_i \to v_j \to t$ for some unique i, j because each u_i (resp. each v_j) has at most 1 unit of in-coming flow and 1 unit of out-going flow.

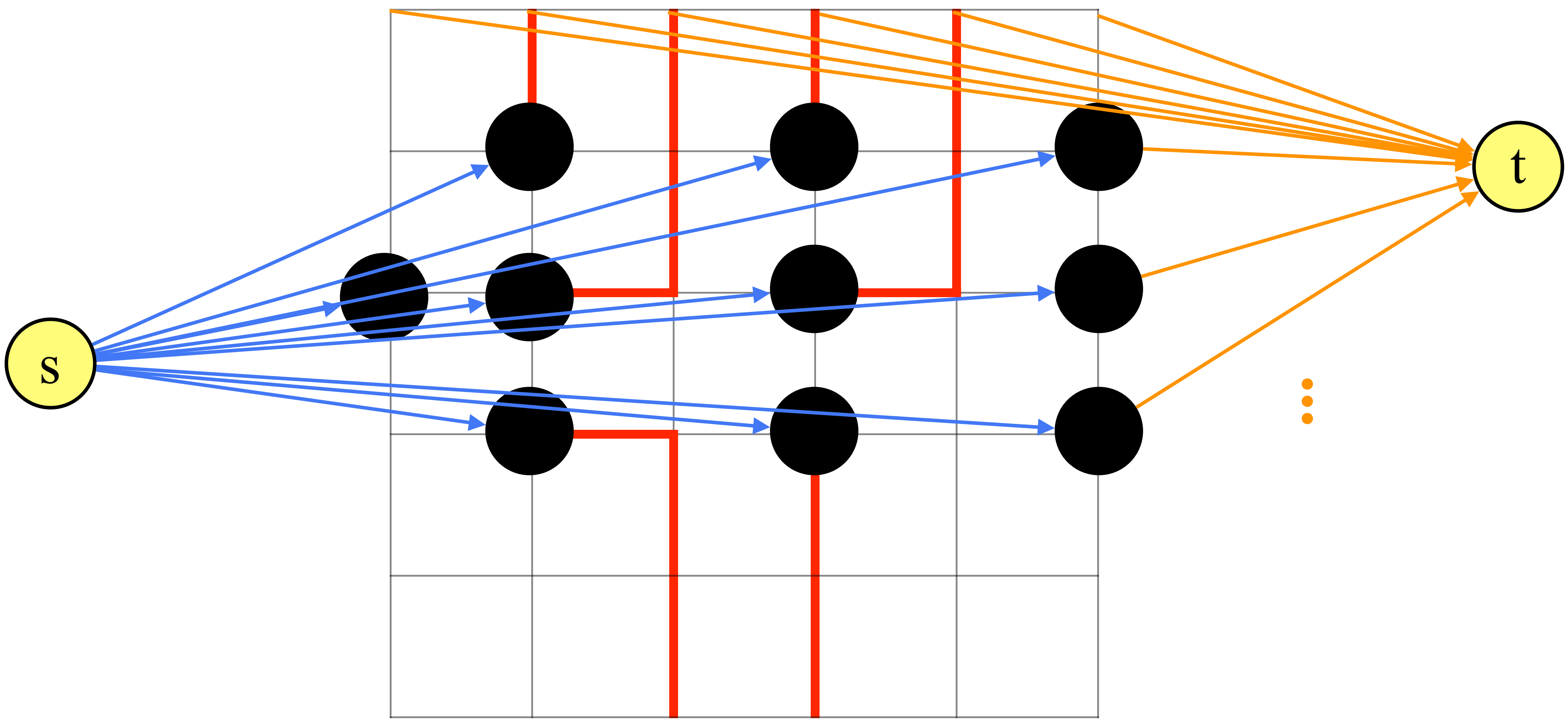
Escape Problem

Escape Problem



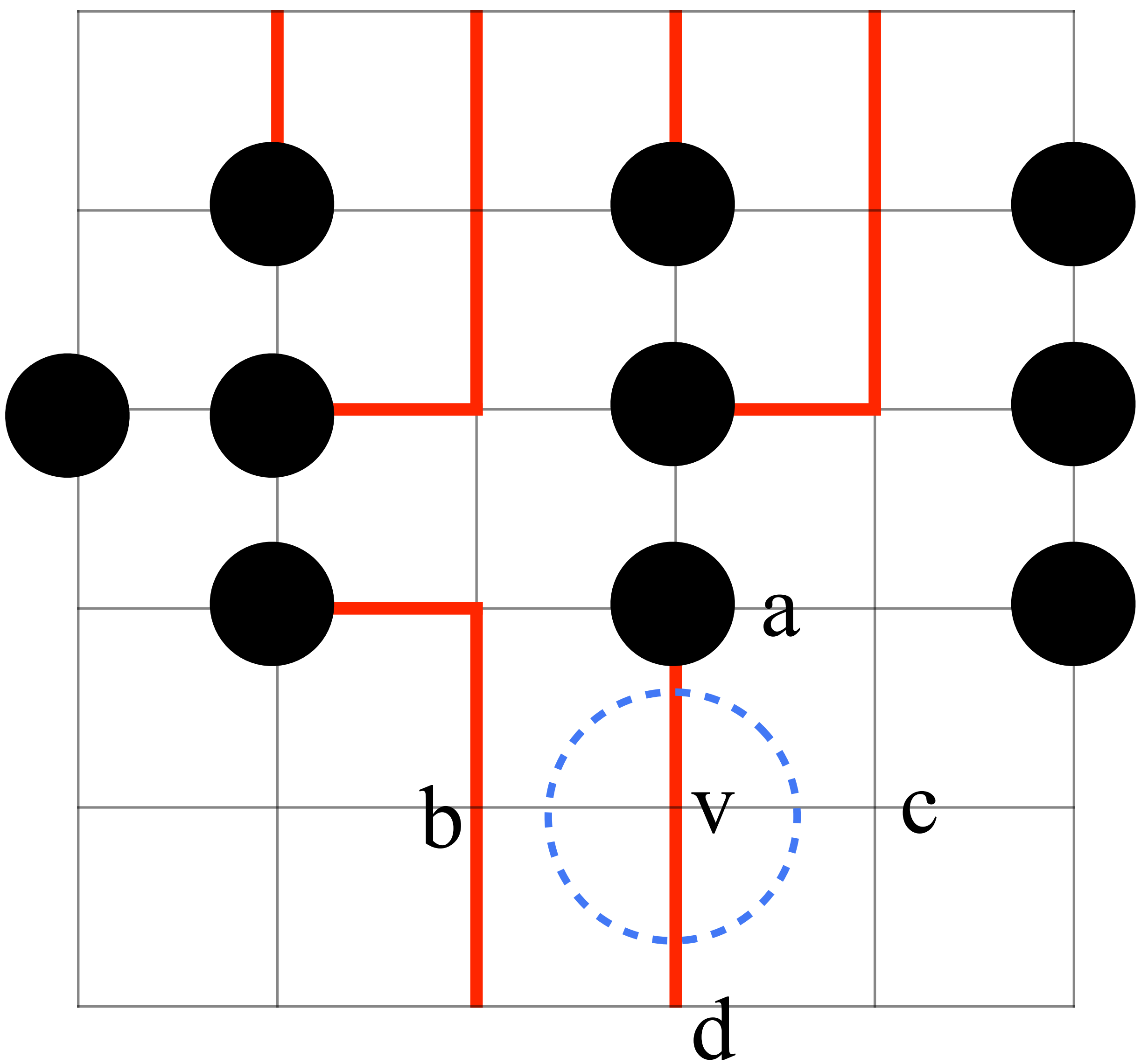
Every black node has to find a escape route from itself to the boundary so that no two routes intersects (vertex disjoint).

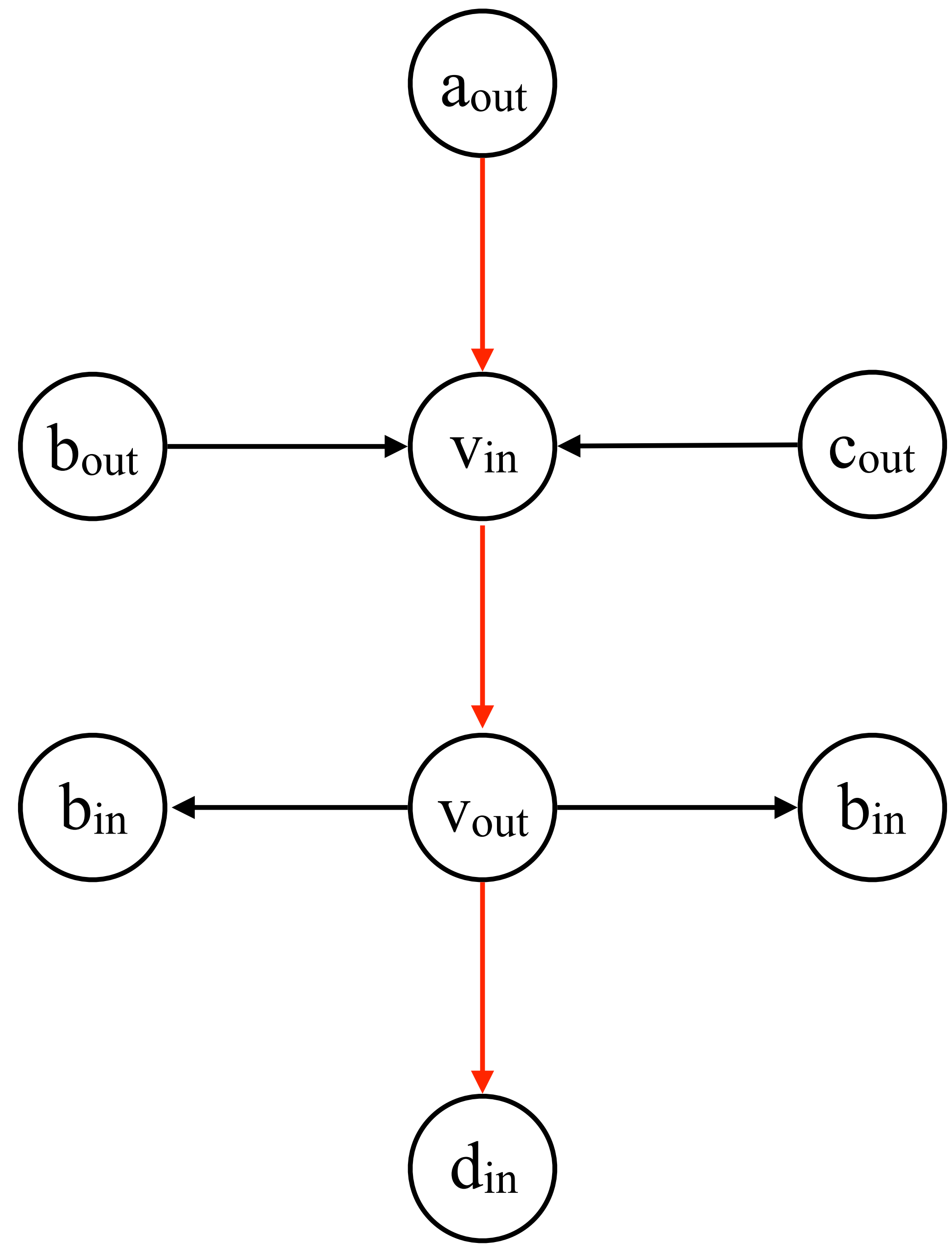
Escape Problem



Every black node has to find a escape route from itself to the boundary so that no two routes intersects (vertex disjoint).

Escape Problem





$c(v_{in}, v_{out}) = 1$ for all v's

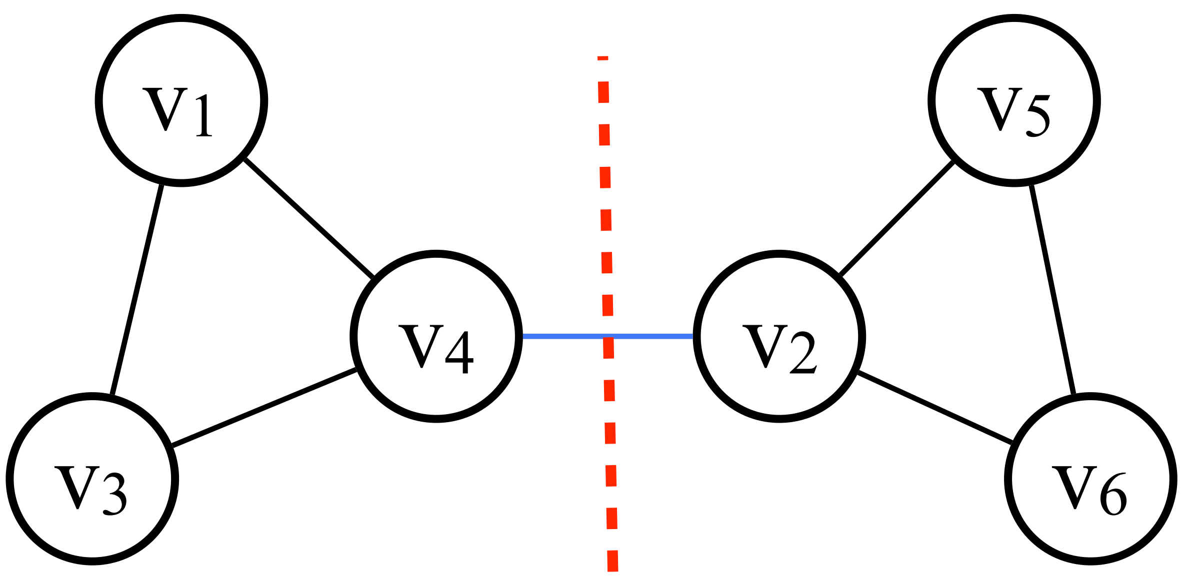
Min Cut

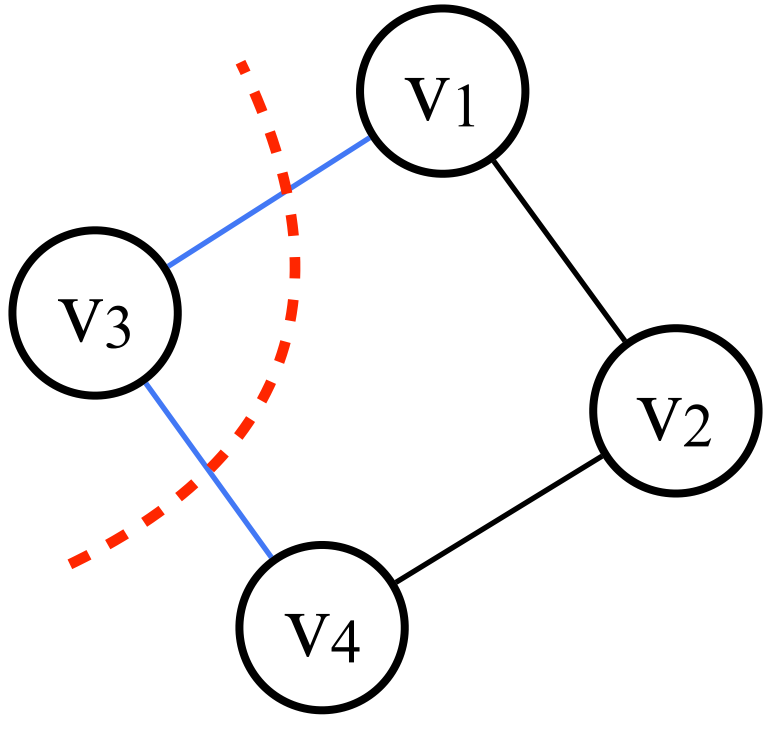
Min Cut

Input: an undirected graph G.

Output: an edge set C whose removal disconnects G so that |C| is minimized.

<u>Example</u>.





Min Cut

```
Min-Cut(G){
    if(G is disconnected){
        return |C| = 0;
    }else{
        let s be an arbitrary node; // a node in one part
        |C| ← ∞;
        foreach(node v in G other than s){
            // guess v to be a node in another part
            find min s-v cut C_sv by computing the max flow
            from s to v; // min-cut max-flow theorem
            if(|C_sv| < |C|) |C| ← |C_sv|;
        }
    }
}
```

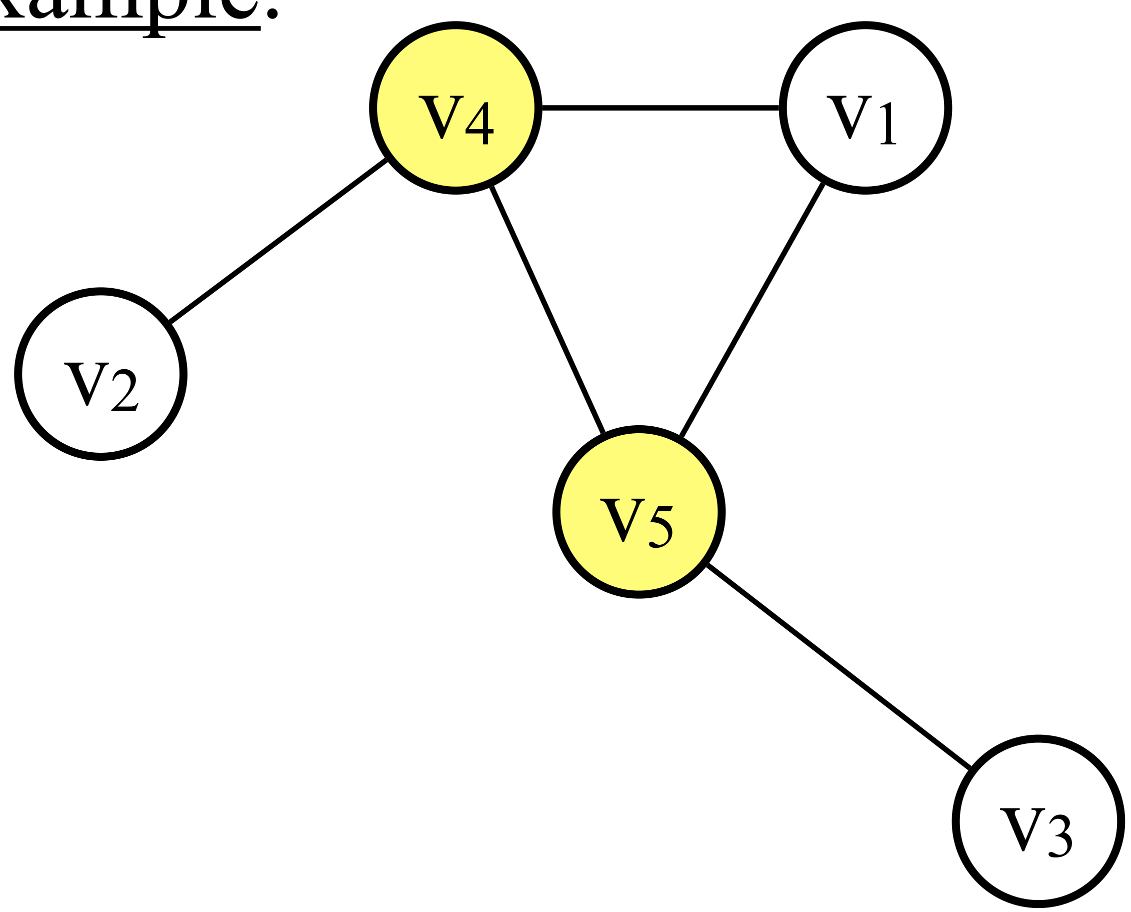
Vertex Cover

Minimum Vertex Cover

Input: an undirected graph G = (V, E).

Output: a subset S of V so that every edge in E has an endpoint in S so that |S| is minimized.

Example.

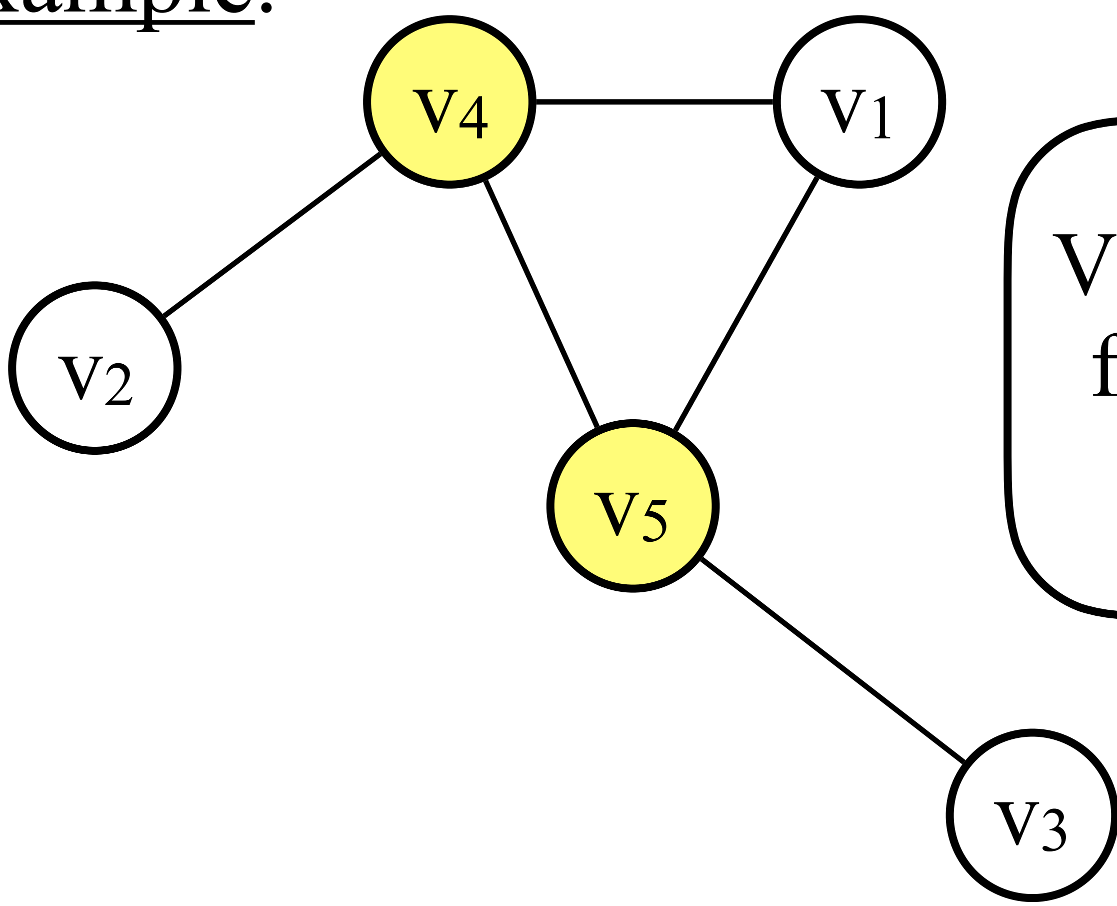


Minimum Vertex Cover

Input: an undirected graph G = (V, E).

Output: a subset S of V so that every edge in E has an endpoint in S so that |S| is minimized.

<u>Example</u>.



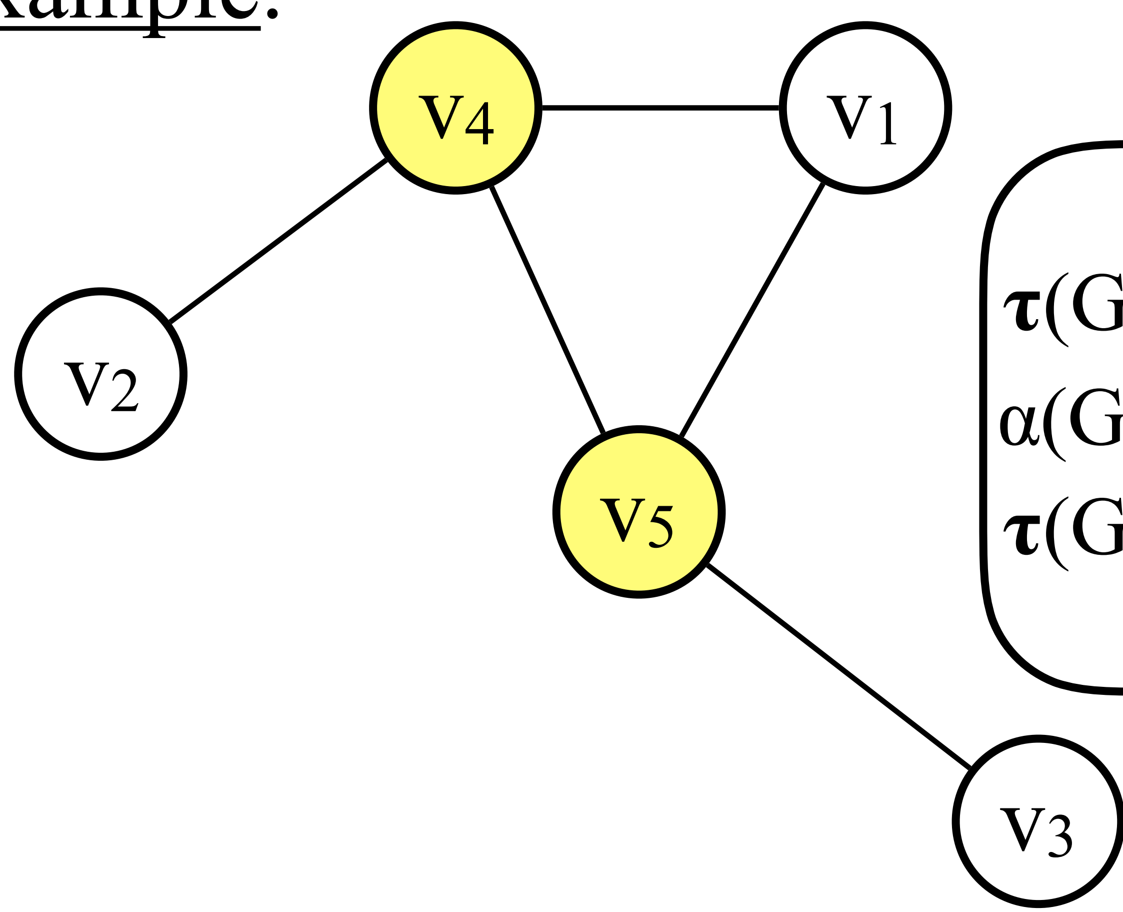
V/S is an independent set; that is, for all edges (u, v) in E, at least one of u, v is not in V/S.

Minimum Vertex Cover

Input: an undirected graph G = (V, E).

Output: a subset S of V so that every edge in E has an endpoint in S so that |S| is minimized.

<u>Example</u>.



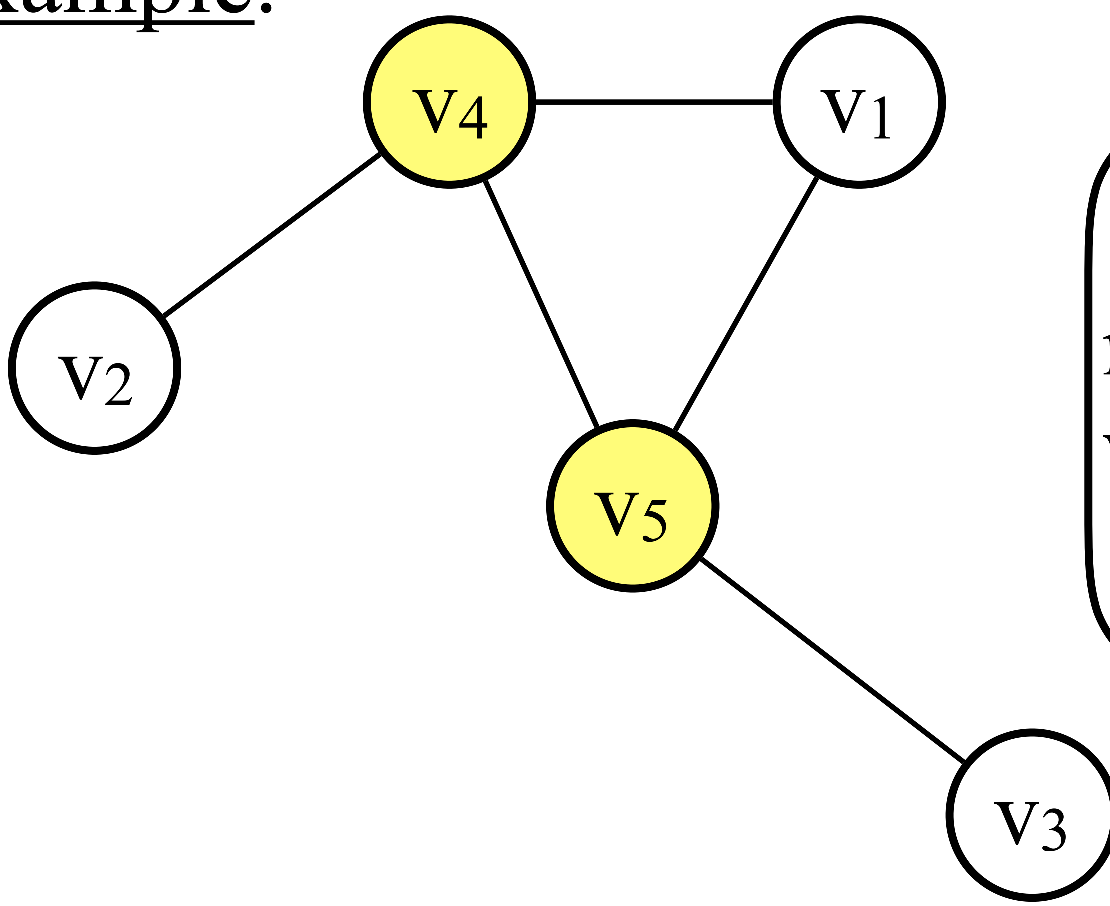
$\boldsymbol{\tau}(G)$ = # nodes in Min Vertex Cover
$\alpha(G)$ = # nodes in Max Independent Set
$\boldsymbol{\tau}(G) + \alpha(G) = V$

Minimum Vertex Cover

Input: an undirected graph G = (V, E).

Output: a subset S of V so that every edge in E has an endpoint in S so that |S| is minimized.

Example.



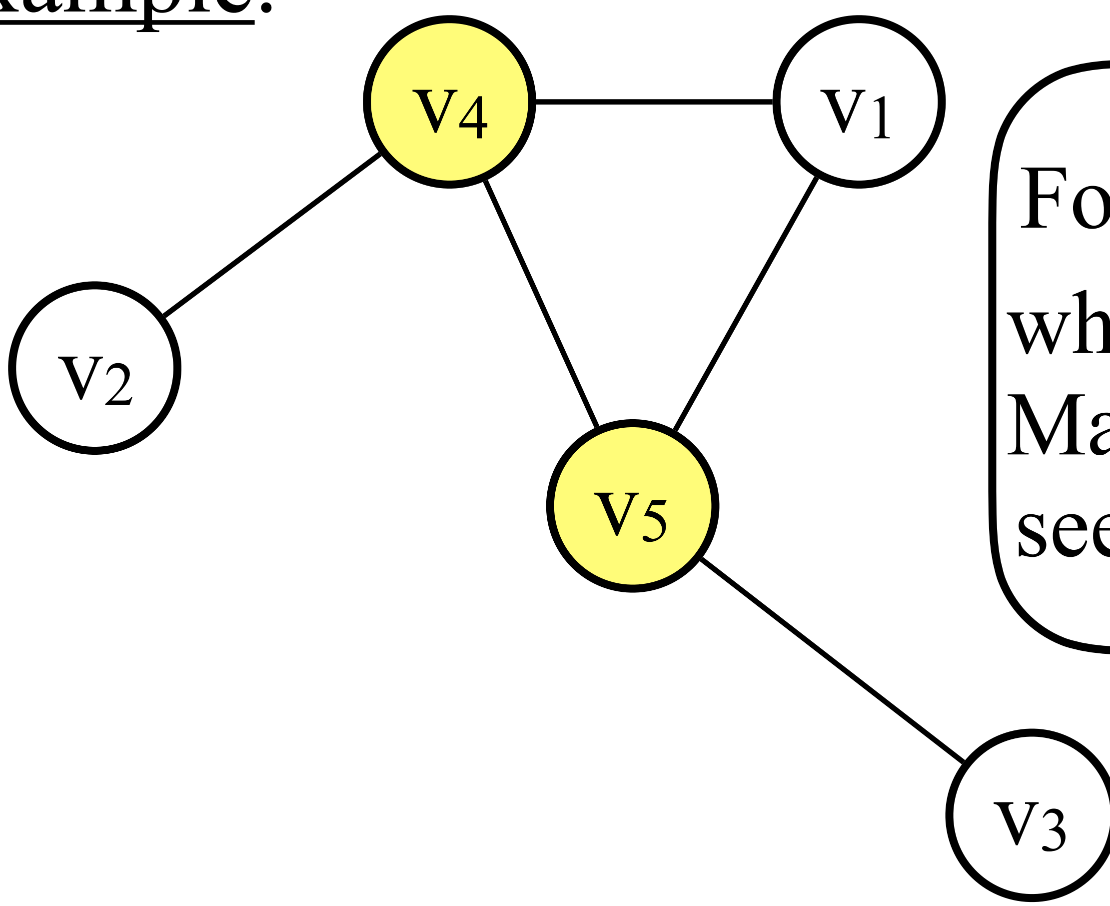
For general graphs, people have no idea how to solve this problem without exhaustive search. We will see this in the lecture of NP.

Minimum Vertex Cover

Input: an undirected graph G = (V, E).

Output: a subset S of V so that every edge in E has an endpoint in S so that |S| is minimized.

Example.



For bipartite graphs, $\boldsymbol{\tau}(G) = \nu(G)$ where $\nu(G)$ be # edges in Maximum Matching. (König Theorem) We will see this in the lecture of LP.

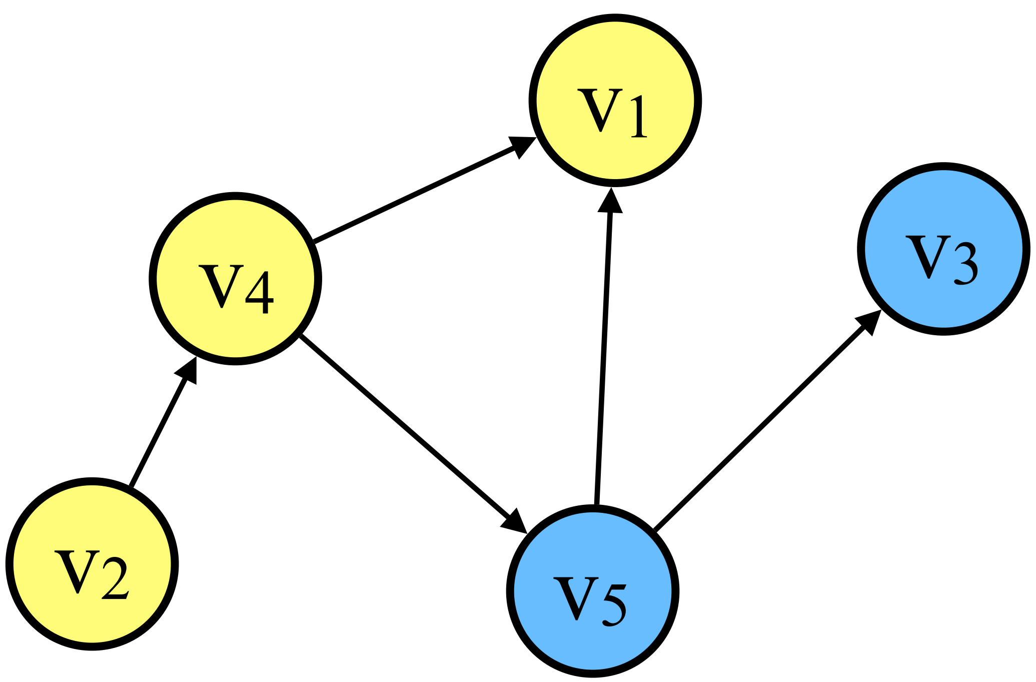
Path Cover

Minimum Path Cover

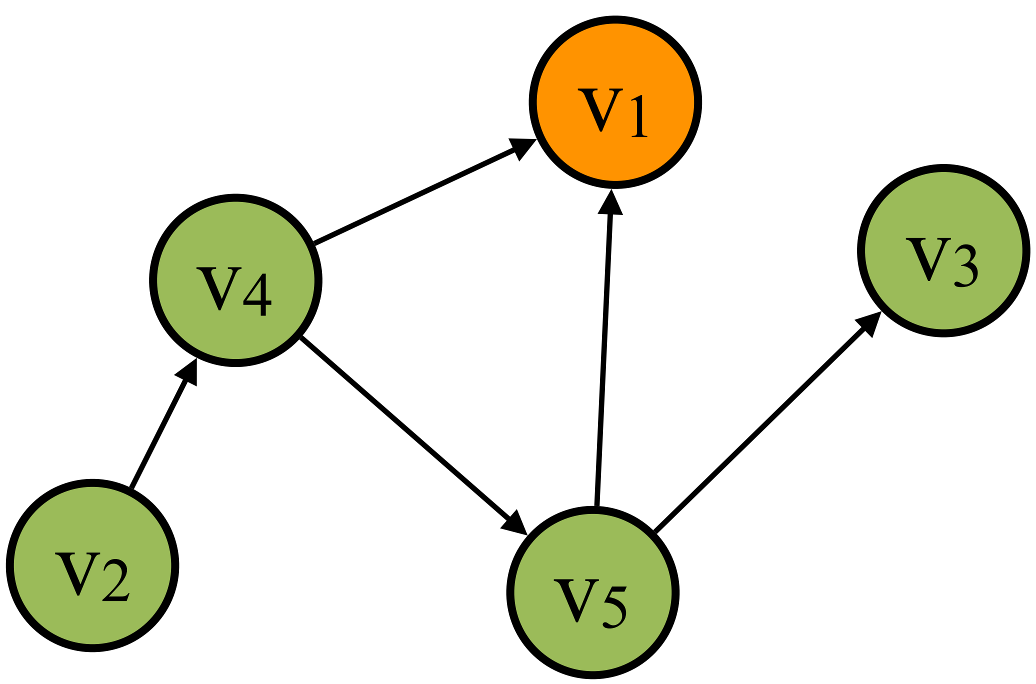
Input: a directed acyclic graph G = (V, E).

Output: a set P of vertex-disjoint directed paths so that |P| is minimized and every node in V is contained in exactly one path in P, noting that paths could have length 0.

Example.

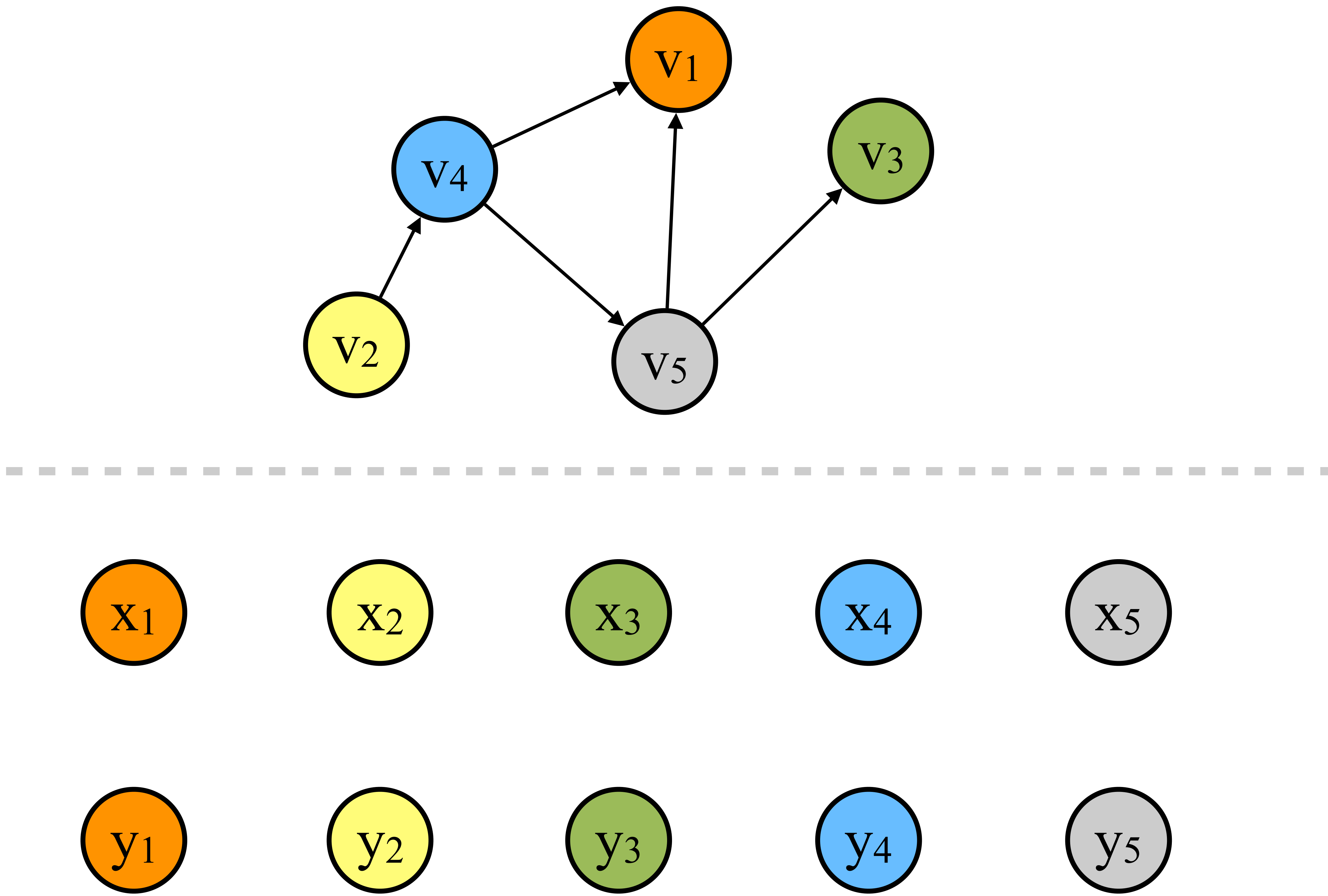


|P| = 2 (optimal)

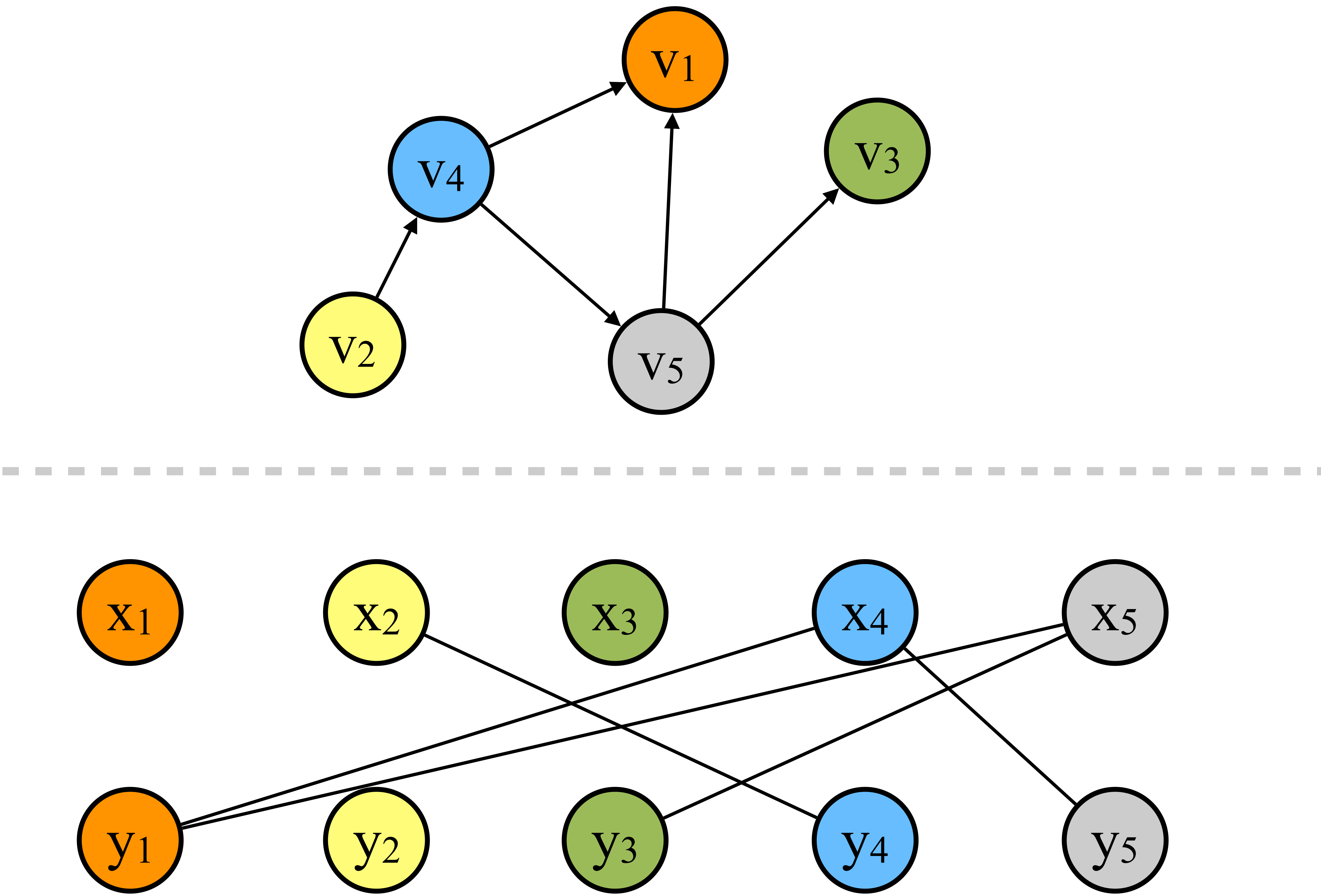


|P| = 2 (optimal)

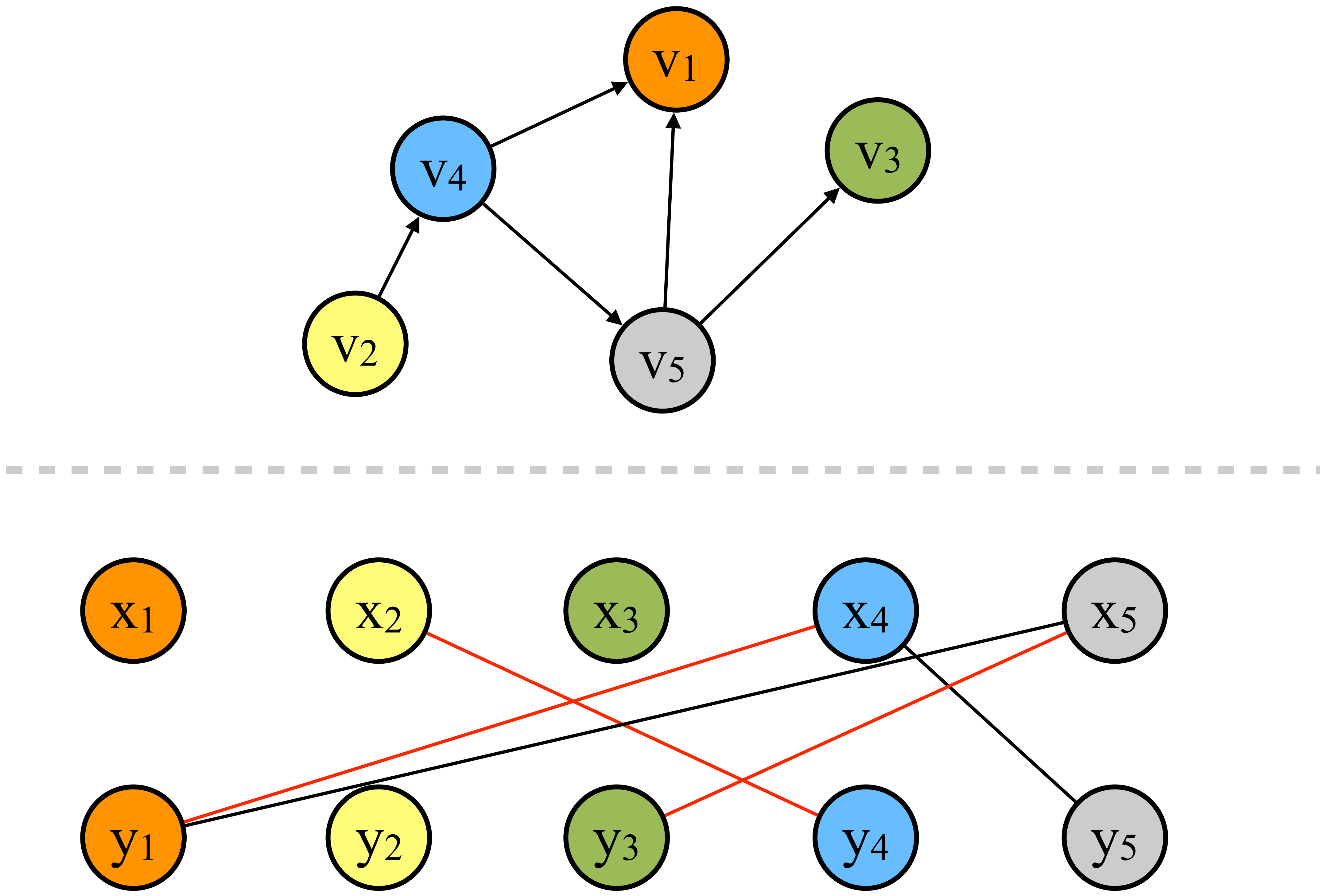
Minimum Path Cover



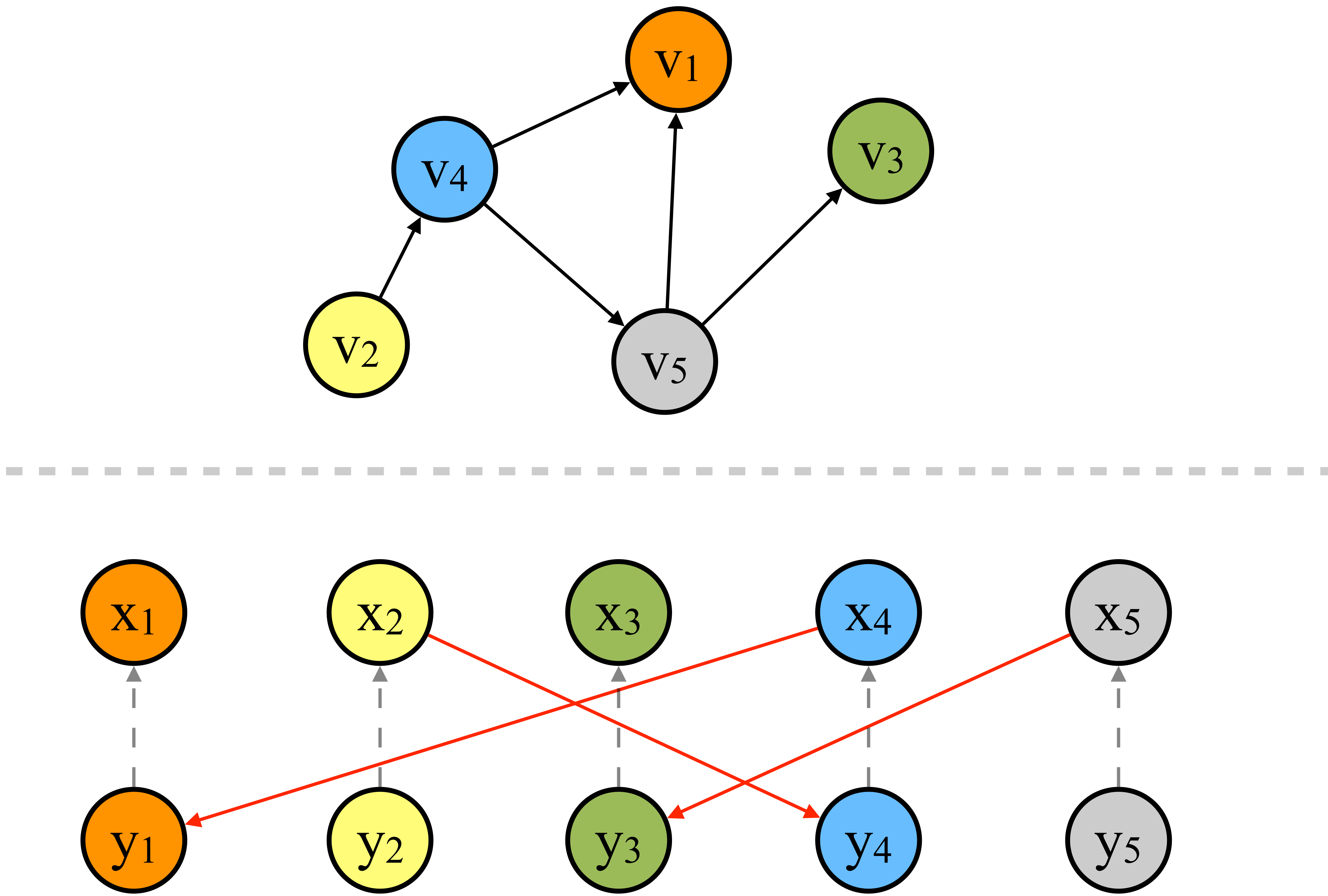
Minimum Path Cover



Minimum Path Cover

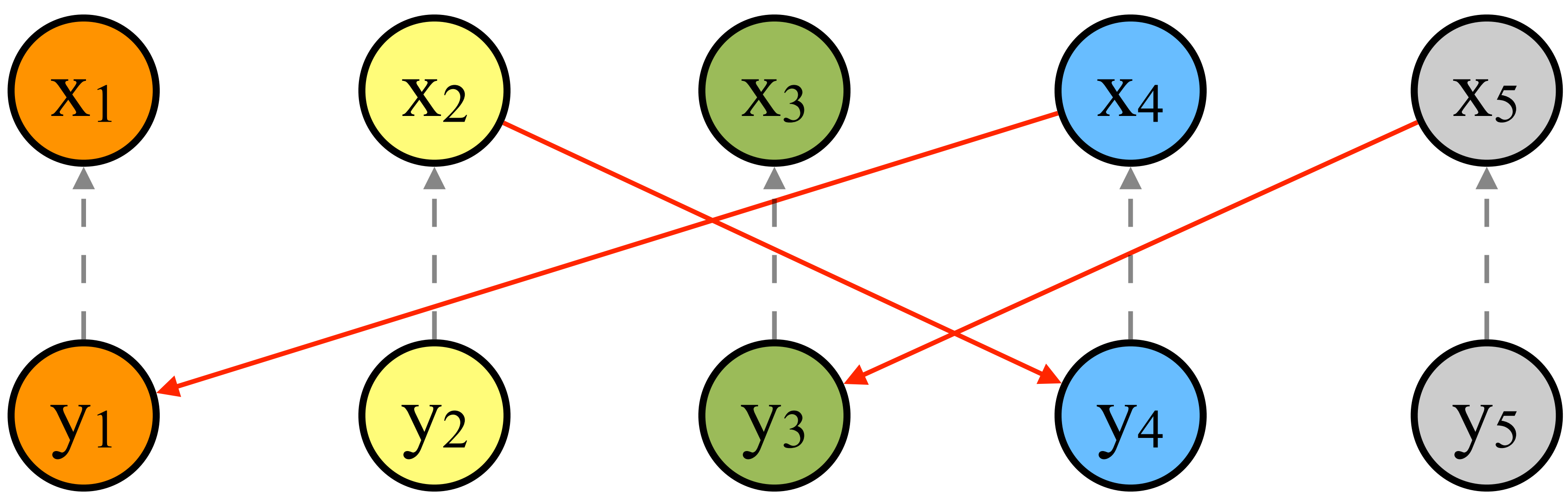


Minimum Path Cover



Minimum Path Cover

The minimized $|P| = n-\nu(G)$ because # edges in a matching = # edges in the corresponding P. If P has k paths, then P has n-k edges. Maximize k ≡ minimize |P|.



Exercise

Can we solve the minimum path cover for any directed graph?