Advanced Algorithms

Lecture 3
Flow Networks and Maximum Flow

Bin Yang
byang@cs.aau.dk

ILO of Lecture 3

- Flow network and maximum flow
  - to understand the formalization of flow networks and flows; and the definition of the maximum-flow problem.
  - to understand the Ford-Fulkerson method for finding maximum flows.
  - to understand the Edmonds-Karp algorithm and to be able to analyze its worst-case running time;
  - to be able to apply the Ford-Fulkerson method to solve the maximum-bipartite-matching problem.

Agenda

- Flow networks, flows, maximum-flow problem
- The Ford-Fulkerson method
- The Edmonds-Karp algorithm
- Maximum-bipartite-matching
- Spatial crowd sourcing

Flow networks

- What if the weights in a weighted graph represent maximum capacities of some flow of material?
  - Capacity: a maximum rate at which the material can flow through.
  - Pipe network to transport fluid (e.g., water, oil)
    - Edges – pipes
    - Vertices – junctions of pipes
  - Data communication network
    - Edges – network connections of different capacities
    - Vertices – routers (do not produce or consume data just move it)
- Concepts (informally):
  - **Source** vertex s (where material is produced).
  - **Sink** vertex t (where material is consumed).
  - For all other vertices – what goes in must go out.
  - Goal: maximum rate of material flow from **source** to **sink**.

Formalization

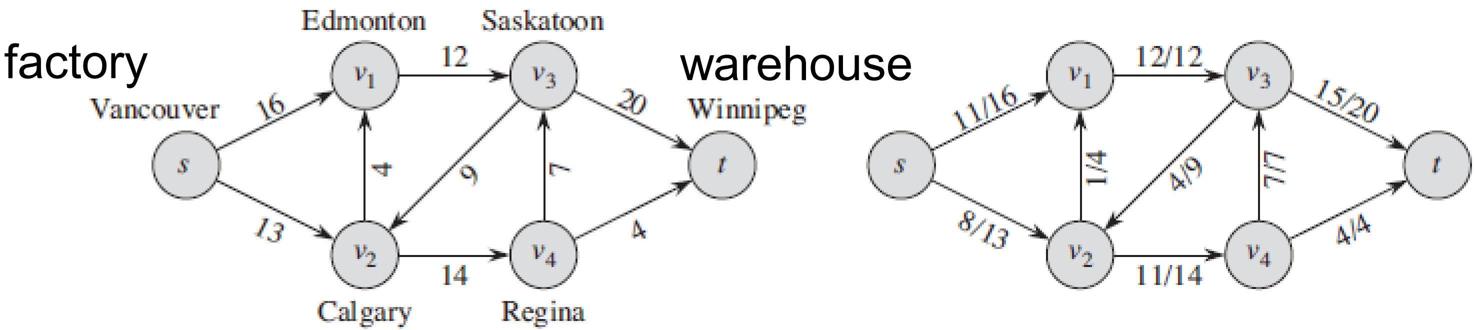
- A **flow network** $G=(V, E)$ is a **directed** graph.
  - Each edge $(u, v) \in E$ has a nonnegative **capacity** $c(u, v) \geq 0$
  - If (u, v) is not in E, then c(u, v)=0.
  - If E contains an edge (u, v), then there is **no** edge (v, u) in the reverse direction.
  - Two special vertices: a ***source*** *s* and a ***sink*** *t.*
  - For any other vertex *v,* there is a path $s \to v \to t$.
- A **flow** in G is a real-valued function f: $V \times V \to R$.
  - ***Capacity constraint***: for all $u, v \in V$, $0 \leq f(u, v) \leq c(u, v)$.
    - Flow from one vertex to another must be nonnegative and must not exceed the given capacity.
  - ***Flow conversation***: for all $u \in V-\{s, t\}$,

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

Flow in equals flow out.

    - Total flow into a vertex other than the source and the sink (i.e., vertex u) must equal to the total flow out of that vertex.

Examples



- Left figure: capacity
- Right figure: flow/capacity, if flow=0, we only denote capacity.

- Edge (s, v_1)
  - $f(s, v_1)=11 < c(s, v_1)=16$
  - Capacity constraint is satisfied.
- V1, which is not the source s and not the sink t.
  - $f(s, v_1)+f(v_2, v_1)=11+1=12$
  - $f(v_1, v_3)=12$
  - Flow conversation is satisfied.

Products cannot be accumulated at intermediate cities, i.e., no warehouses at intermediate cities.

Maximum-flow problem

- Consider the source s.
- The **value** of flow f, denoted as |f|, is defined as

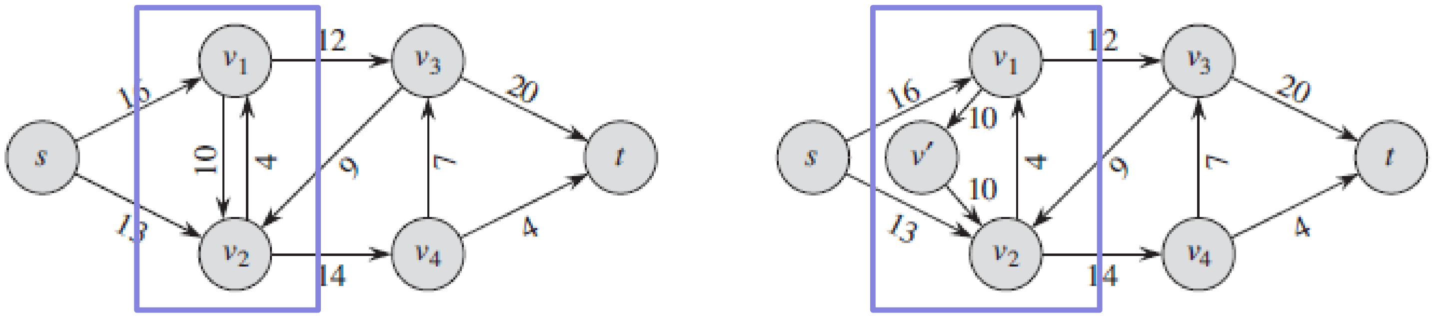
$$|f| = \sum_{v \in V} f(s,v) - \boxed{\sum_{v \in V} f(v,s)} \qquad 0$$

  - Total flow out of the source minus the flow into the source.
  - Typically, a flow network will not have any edges into the source, and the flow into the source will be zero.

- Maximum-flow problem:
  - Given a flow network G with source s and sink t, we wish to find a flow of maximum value.

Anti-parallel edges

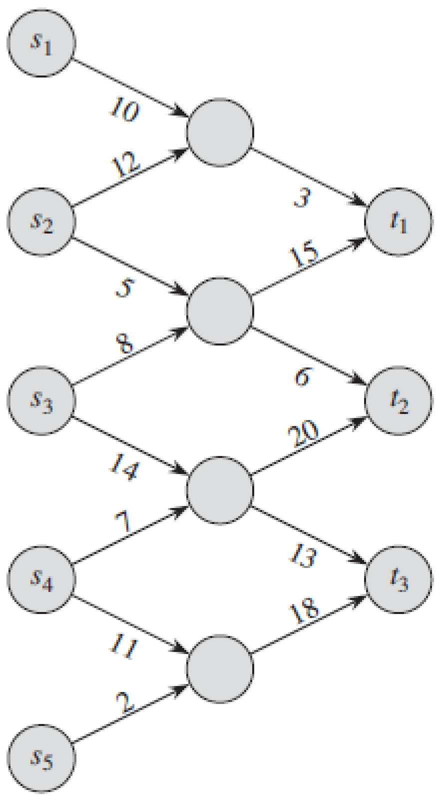
- To simplify the discussion we do not allow both (u, v) and (v, u) together in the graph.
  - If E contains an edge (u, v), then there is no edge (v, u) in the reverse direction.
- Easy to eliminate such antiparallel edges by introducing artificial vertices.

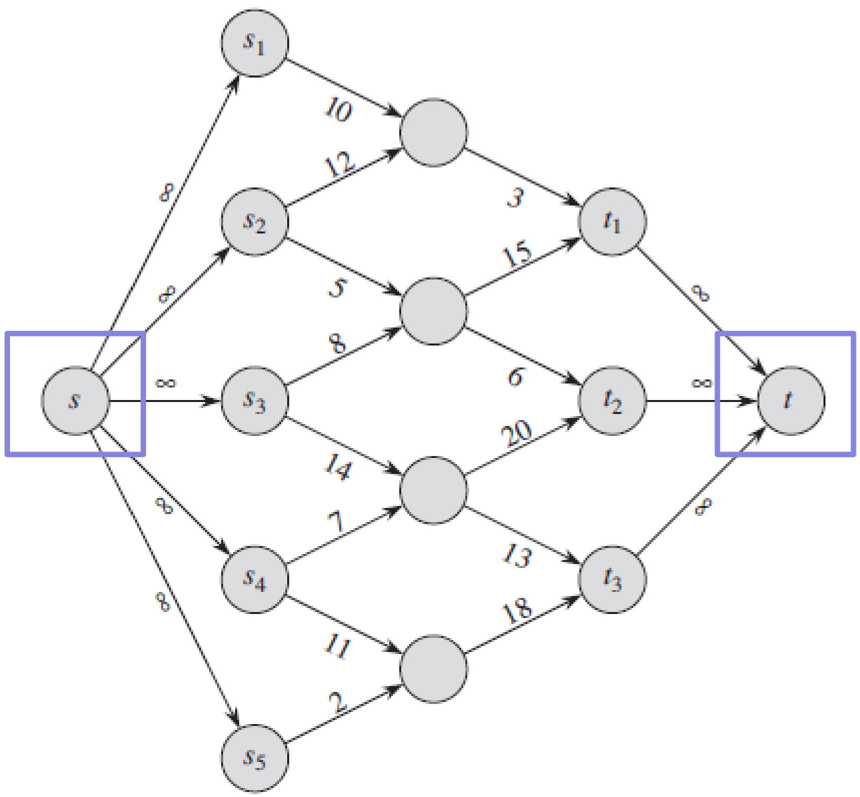


  - Antiparallel edges: (v_1, v_2) and (v_2, v_1)
  - Choose one of the two antiparallel edges, e.g., (v_1, v_2), split it by adding a new vertex v’, and replace (v_1, v_2) by (v_1, v’) and (v’, v_2).
  - Set the capacity of the two new edges to the capactity of the original edge.

Multiple sources and multiple sinks

- Example: multiple factories and multiple warehouses.
- Introducing a super-source s and super-sink t.
  - Connect s to each of the original source s_i and set its capacity to ∞.
  - Connect t to each of the original sink t_i and set its capacity to ∞.





Agenda

- Flow networks, flows, maximum-flow problem
- The Ford-Fulkerson method
- The Edmonds-Karp algorithm
- Maximum-bipartite-matching
- Spatial crowd sourcing

The Ford-Fulkerson method

- A method, but not an algorithm
  - It encompasses several implementations with different running times.
- The Ford-Fulkerson method is based on
  - Residual networks
  - Augmenting paths

Residual networks

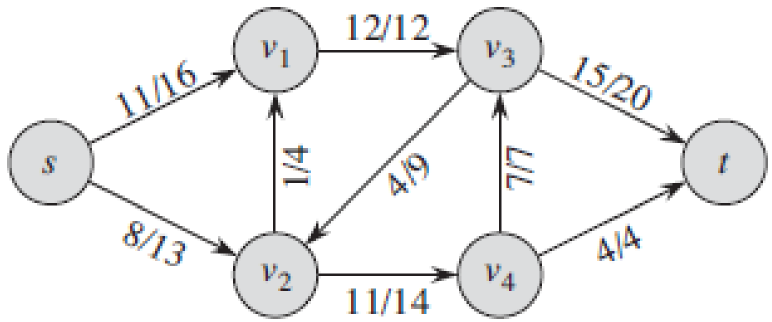
- Given a flow network G and a flow f, the residual network G_f consists of edges whose ***residual capacities*** are greater than 0.
  - Formally, $G_f=(V, E_f)$, where $E_f=\{(u, v) \in V\times V: c_f(u, v)>0\}$.
- ***Residual capacities***:

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

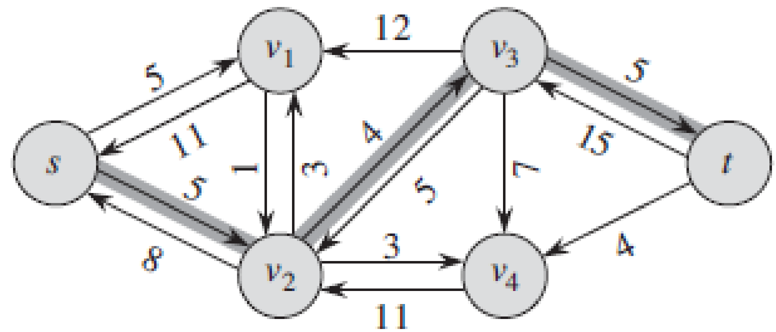
  - The amount of additional flow that can be allowed on edge (u, v).
  - The amount of flow that can be allowed on edge (v, u), i.e., the amount of flow that can be canceled on the opposite direction of edge (u, v).

Example

Flow on a flow network



Residual network



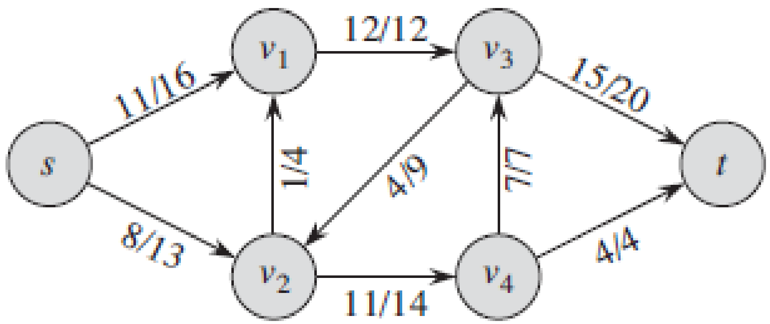
- $c_f(s, v_1)=c(s, v_1)-f(s, v_1)=16-11=5$
- $c_f(v_1, s)=f(s, v_1)=11$
- $c_f(v_1, v_3)=c(v_1, v_3)-f(v_1, v_3)=12-12=0$. Thus, edge (v_1, v_3) is not in G_f.
- $c_f(v_3, v_1)=f(v_1, v_3)=12$.
- …

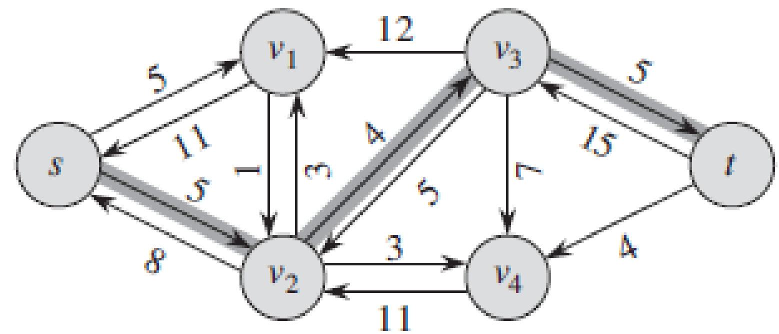
The edges in the residual network G_f are either edges in E or their reversals:
$|E_f| \leq 2|E|$

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

Augmenting paths

- Given a flow network G and a flow f, an ***augmenting path*** *p* is a simple path from s to t in the ***residual network*** G_f.





  - p=<s, v_2, v_3, t>

- ***Residual capacity*** of an augmenting path *p*:
  - How much additional flow can we send through an augmenting path?
  - $c_f(p)=\min\{c_f(u, v): (u, v) \text{ is on path } p\}$
  - $c_f(p)=\min\{5, 4, 5\}=4$
  - The edge with the minimum capacity in *p* is called ***critical*** edge.
    - (v2, v3) is the critical edge of *p.*

Augmenting a flow

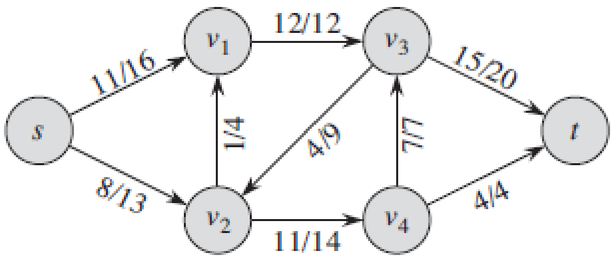
- Given an augmenting path p, we define a flow f_p on the residual network G_f.

$$f_p(u,v) = \begin{cases} c_f(p) & \text{if } (u,v) \text{ is on } p\ , \\ 0 & \text{otherwise}\ . \end{cases}$$

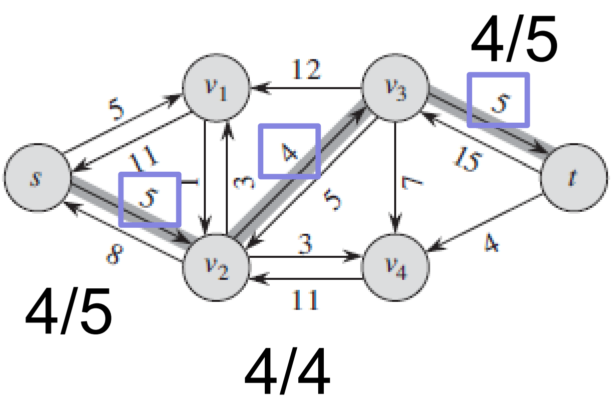
  - The flow value of $|f_p|=c_f(p)>0$.
- If f is a flow in G and f_p is a flow in the corresponding residual network G_f, we define $f\uparrow f_p$, the augmentation of flow f by f_p, to be a function from V×V to R.
  - $f\uparrow f_p(u, v)=$
    - $f(u, v) + f_p(u, v)-f_p(v, u)$ if $(u, v)\in E$,
    - 0 otherwise.
- $f\uparrow f_p$ is also a flow in G with value $|f\uparrow f_p| = |f| +|f_p| > |f|$.
  - By augmenting a flow by the flow of an augmenting path, we get a new flow with greater flow value.

Examples

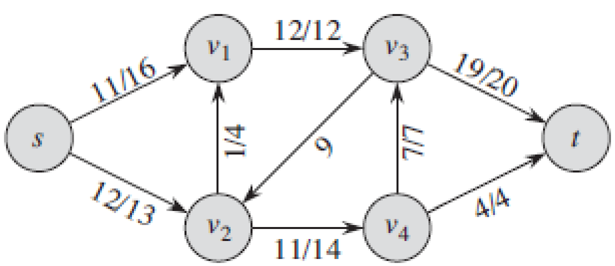
$f \uparrow f_p(u, v) = f(u, v) + f_p(u, v) - f_p(v, u)$



- Original flow f, with flow value $|f| = 11 + 8 = 19$

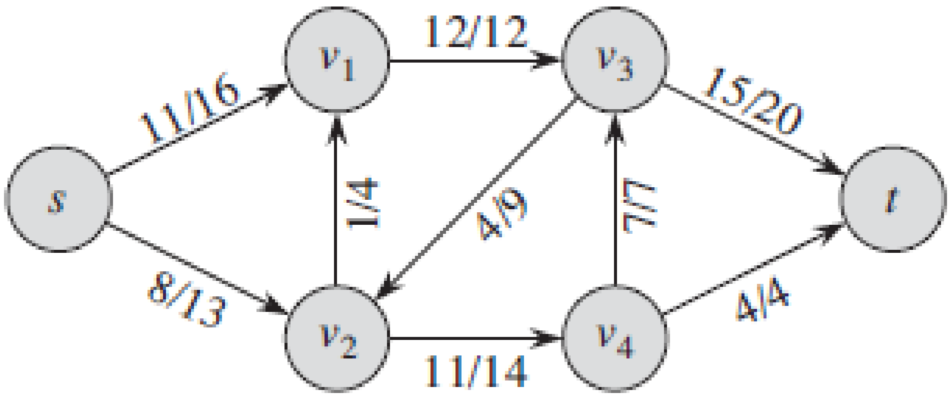


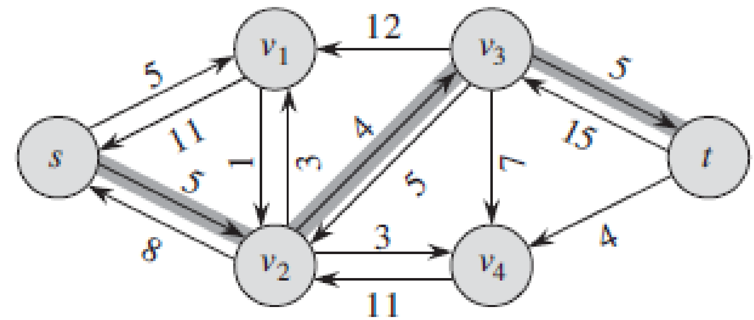
- Augmenting path *p* on the residual network. Flow f_p based on the augmenting path is with flow value 4.
  - $f_p(s, v_2) = f_p(v_2, v_3) = f_p(v_3, t) = 4$
  - $|f_p| = 4$



- Augment f by f_p
  - $f \uparrow f_p(s, v_2) = 8 + 4 - 0 = 12$
  - $f \uparrow f_p(v_3, v_2) = 4 + 0 - 4 = 0$
  - $f \uparrow f_p(v_3, t) = 15 + 4 - 0 = 19$
- New flow value: $|f \uparrow f_p| = 11 + 12 = 23$
- $|f| + |f_p| = 19 + 4 = 23$

Mini-quiz (also on Moodle)





- Compute the corresponding residual network.
- What is the residual capacity of augmenting path p=<s, v1, v2, v3, t>?
  - 1
- What is the new flow value by augmenting the current flow by the flow of p=<s, v1, v2, v3, t>?
  - 20

Exercises this week

- Simon is unavailable this week.
- Razvan will replace Simon.

The Ford-Fulkerson method

```
Ford-Fulkerson(G,s,t)
01 for each edge (u,v) ∈ G.E do
02     f(u,v) ← 0
03 while there exists a path p from s to t in residual
   network G_f do
04     c_f = min{c_f(u,v): (u,v) ∈ p}
05     for each edge (u,v) ∈ p do
06         if (u,v) ∈ G.E then f(u,v) ← f(u,v) + c_f
07         else f(v,u) ← f(v,u) - c_f
08 return f
```

Initialize a flow with flow value 0.

Get critical edge and residual capacity

Augment the existing flow by the flow of the augmenting path

$$f\uparrow f_p(u, v)= f(u, v) + f_p(u, v)-f_p(v, u)$$

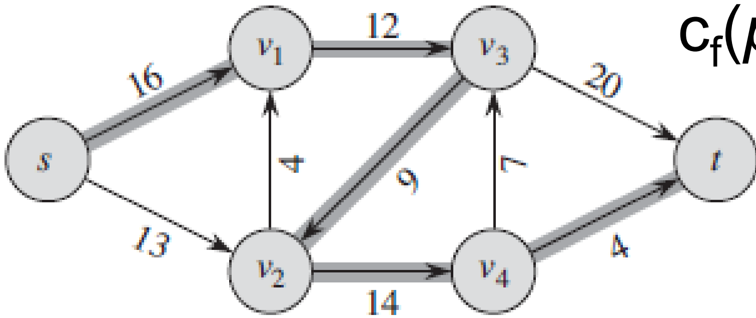
- 1. Find an augmenting path in the residual network.
- 2. Augment the existing flow by the flow of the augmenting path.
- 3. Keep doing this until no augmenting path exists in the residual network.

- The algorithms based on this method differ in how they choose *p* in line 3.
- Correctness is provided by the ***Max-flow min-cut*** theorem.

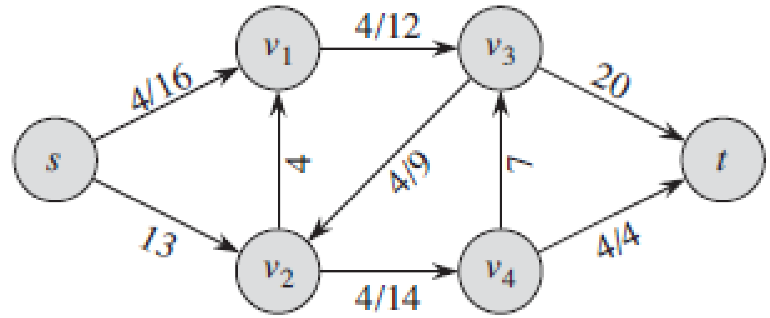
Example

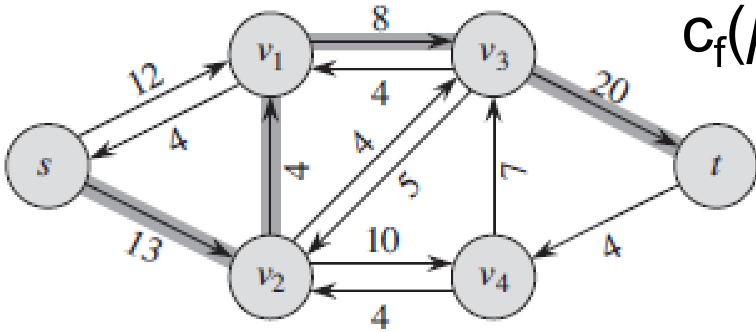
Residual network

New Flow

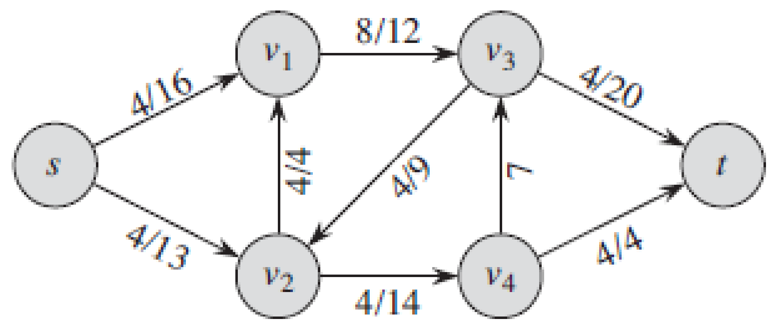


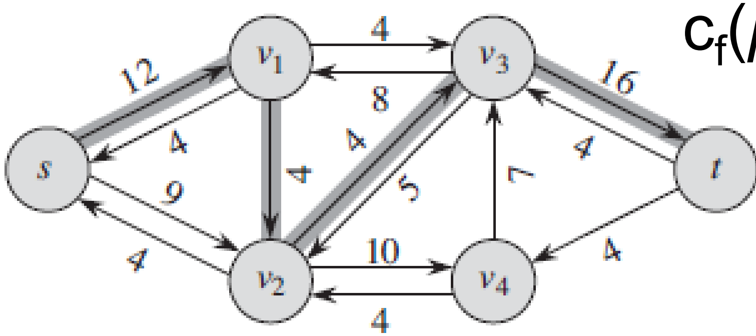
$c_f(p)$=4



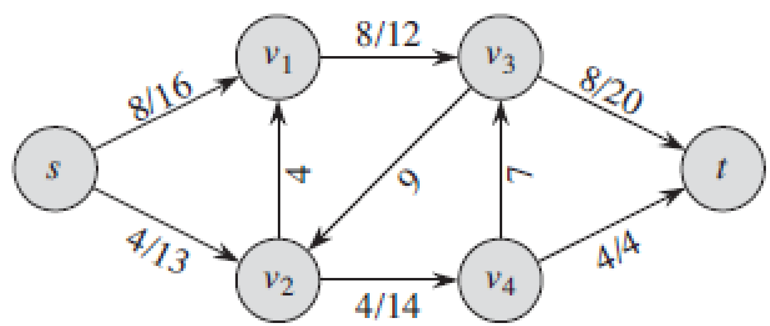


$c_f(p)$=4





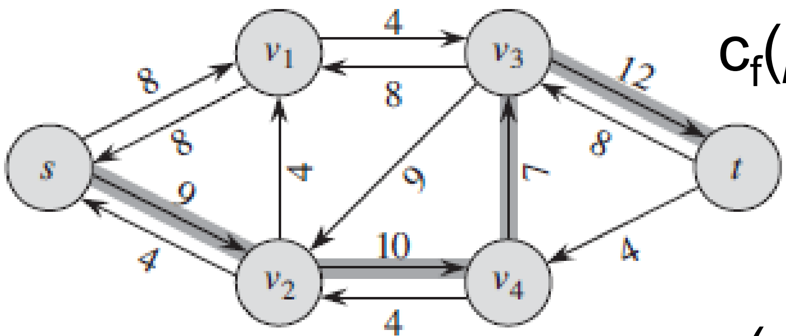
$c_f(p)$=4



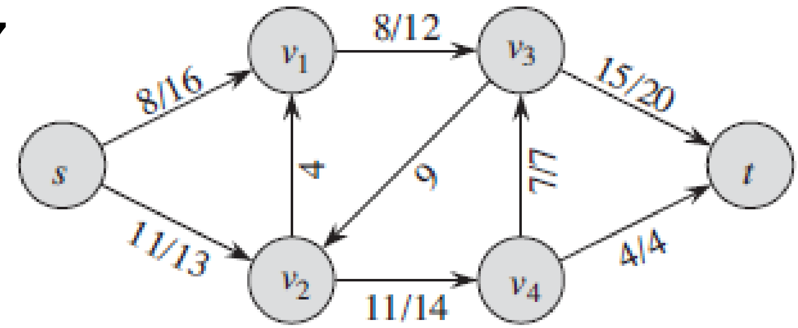
Example 2

Residual network

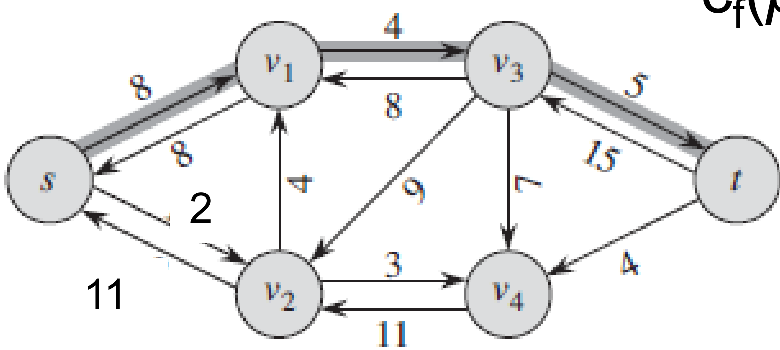
New Flow

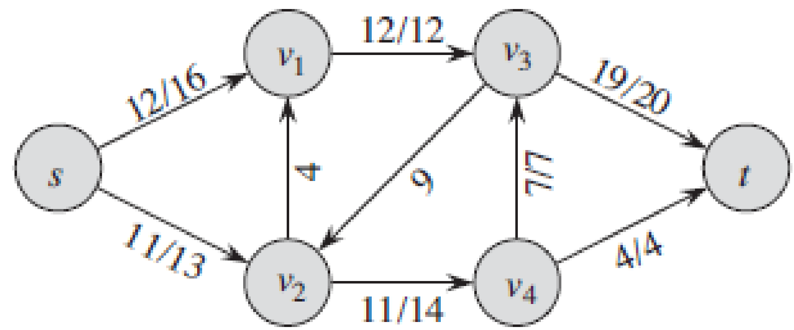


$c_f(p)=7$

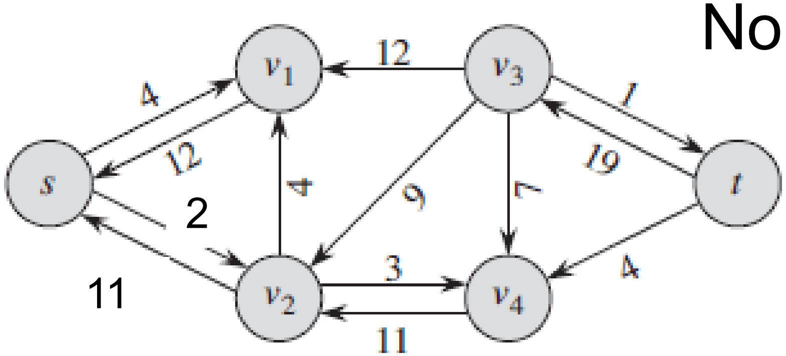


$c_f(p)=4$





No augmenting path anymore



Maximum flow: 12+11=23

Correctness of Ford-Fulkerson

- Why this method is correct?
- How do we know that when the method terminates, i.e., when there are no more augmenting paths, we have actually find a maximum flow?

- Max-flow min-cut theorem

Cuts

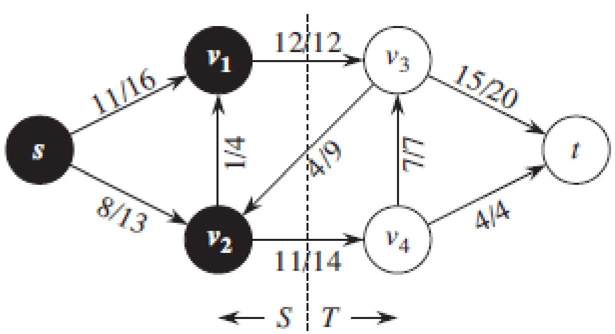
- A ***cut*** is a partition of V into S and T=V-S, such that **s** ∈ **S** and **t** ∈ **T**.
- The ***net flow*** f(S, T) across the cut (S, T) is defined as

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

  - The flow going from S to T minus the flow going from T to S.
- The ***capacity*** c(S, T) of the cut (S, T) is defined as

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

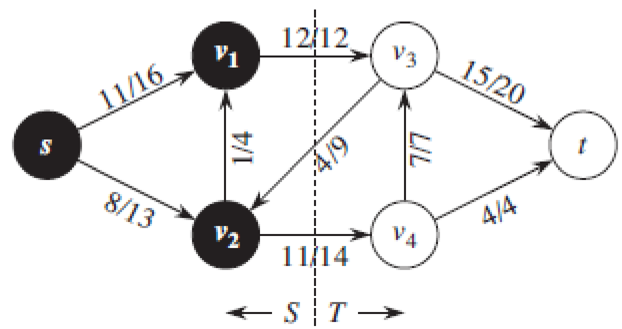
  - The sum of the capacities of edges going from S to T.



Black/White vertices are in S/T.
f(S, T)=f(v1, v3)+f(v2, v4)-f(v3, v2)
=12 + 11 – 4 =19.
c(S, T)=c(v1, v3)+c(v2, v4)
=12+14=26.

Minimum cut

- Minimum cut
  - A cut whose capacity is minimum over all cuts of the network.
- Given a flow f in G, for any cut (S, T) on G, we have that the net flow across (S, T) is same with the value of the flow, i.e., |f|.
  - |f|=f(S, T)



f(S, T)=|f|=19

c(S, T)=12+14=26

- The value of any flow f in G is bounded by the capacity of any cut of G.
  - |f|≤C(S, T)
- The maximum flow is bounded by the capacity of the minimum cut.
  - We cannot deliver more than the bottleneck allows.

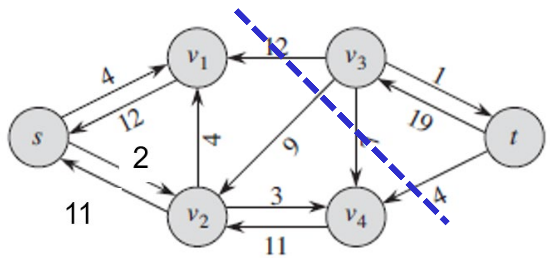
Max-flow min-cut theorem

- If *f* is a flow in G, the following conditions are equivalent:
  - 1. *f* is a maximum flow;
  - 2. The residual network G_f contains no augmenting paths.
  - 3. |f|=c(S, T) for some cut (S, T) of G.

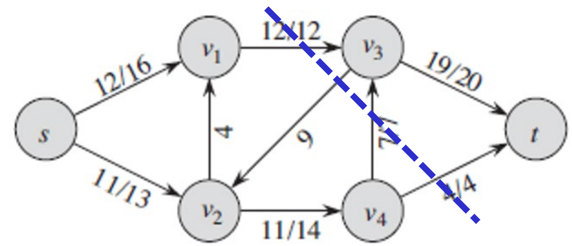
- The correctness of Ford-Fulkerson method.
  - 2=>1
  - We prove 2=>3 and then 3=>1

2 => 3

- 2. The residual network G_f contains no augmenting paths.
- 3. $|f|=c(S, T)$ for some cut (S, T) of G.



Residual network G_f

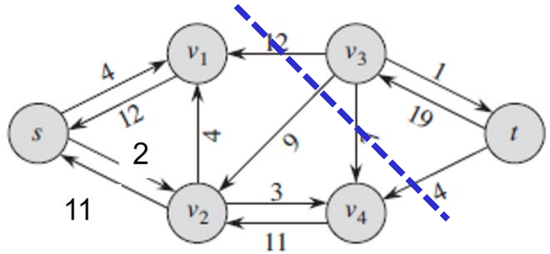


Corresponding flow f on network G

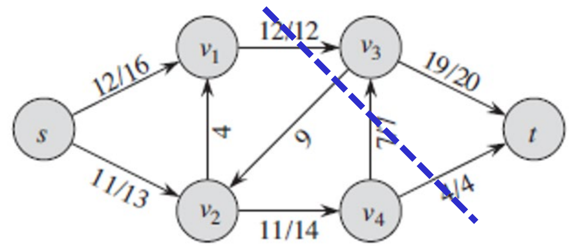
- Let S includes vertices that are reachable from s, and T includes the remaining vertices.
  - S={s, v1, v2, v4}, T={t, v3}
- Consider vertex u that belongs to S and vertex v that belongs T
  - Case 1:
    - If (u, v) is an edge in G, we must have f(u, v) = c(u, v). E.g., (v_1, v_3).
    - Otherwise, (u, v) should appear in G_f and thus make v belong to S.
  - Case 2:
    - If (v, u) is an edge in G, we must have f(v, u)=0. E.g., (v_3, v_2).
    - Otherwise, (u, v) should appear in G_f and thus make v belong to S.

2 => 3

- 2. The residual network G_f contains no augmenting paths.
- 3. $|f|=c(S, T)$ for some cut (S, T) of G.



Residual network G_f



Corresponding flow f on network G

A flow equals to the net flow of any cut.

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

Case 1 *Case 2*

$$= \sum_{u\in S}\sum_{v\in T} c(u,v) - \sum_{v\in T}\sum_{u\in S} 0$$

$$= c(S,T)$$

3 => 1

- 3. |f|=c(S, T) for some cut (S, T) of G.
- 1. *f* is a maximum flow;

- We know that |f|≤c(S, T) for all cuts (S, T)
  - We cannot deliver more than the bottleneck allows.
  - When |f|=c(S, T), this means |f| is a maximum flow.
    - If there exists an even larger flow value |f'| > |f|, then |f'| is also larger than c(S, T), which contradicts that all flows should be no larger than the capacity of any cut.

Worst-case running time

```
Ford-Fulkerson(G,s,t)
for each edge (u,v) ∈ G.E do
    f(u,v) ← 0
while there exists a path p from s to t in residual
  network G_f do
    c_f = min{c_f(u,v): (u,v) ∈ p}
    for each edge (u,v) ∈ p do
        if (u,v) ∈ G.E then f(u,v) ← f(u,v) + c_f
        else f(v,u) ← f(v,u) - c_f
return f
```

Initialize a flow with flow value 0.
θ(E)

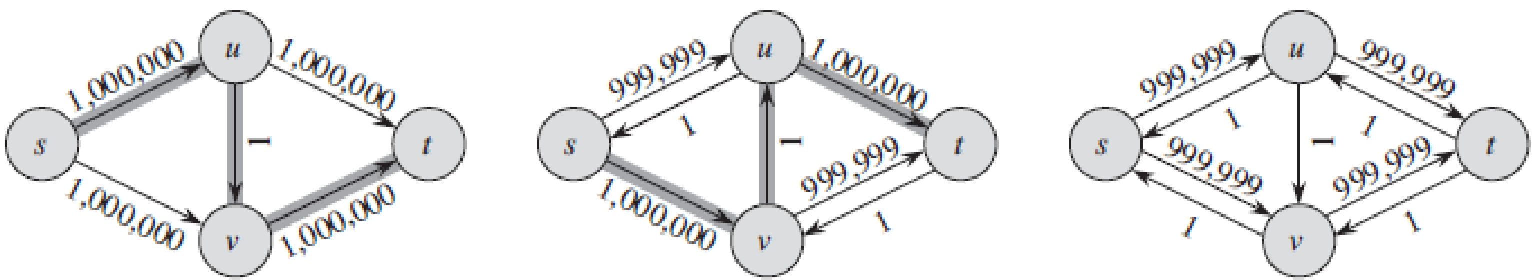
The inner loop:
Find an augmenting path p and augment current flow by the flow of the augmenting path.
O(E)

Outer loop: assume that the while loop iterates x times.

In total, we have O(xE)

Worst-case running time

- Assume integer flows: capacities are integer values.
  - Appropriate scaling transformation can transfer rational numbers to integral numbers.
- Each augmentation increases the value of the flow by some positive amount.
  - Worst case: each time the flow value increases by 1.



(a) (b) (c)

- s, u, v, t
- s, v, u, t
- s, u, v, t
- ….

Worst-case running time

- Identifying the augmenting path and augmentation can be done in $O(E)$.
- Total *worst-case* running time $O(E\,|f^*|)$, where f^* is the max-flow found by the algorithm.
- *Lesson: how an augmenting path is chosen is very important!*

Agenda

- Flow networks, flows, maximum-flow problem
- The Ford-Fulkerson method
- The Edmonds-Karp algorithm
- Maximum-bipartite-matching
- Spatial crowd sourcing

The Edmonds-Karp algorithm

- In line 3 of Ford-Fulkerson method, the Edmonds-Karp treats the residual network as an un-weighted graph and finds the shortest path as an augmenting path.
  - Finding the shortest path in an un-weighted graph is done by calling breath first search (BFS) from source vertex s.

BFS

```
BFS(G,s)
for each vertex a ∈ G.V()
    a.setcolor(white)
    a.setd(∞)
    a.setparent(NIL)
s.setcolor(gray)
s.setd(0)
Q.init()
Q.enqueue(s)
while not Q.isEmpty()
    a ← Q.dequeue()
    for each b ∈ a.adjacent() do
        if b.color() = white then
            b.setcolor(gray)
            b.setd(a.d() + 1)
            b.setparent(a)
            Q.enqueue(b)
    a.setcolor(black)
```

Initialize all vertices: Θ(|V|)

Insert s to a queue Q.
Constant time

Each vertex is enqueued and dequeued ***at most*** once (only when it is white). Assume de-(en-)queue is O(1), then in total O(|V|).

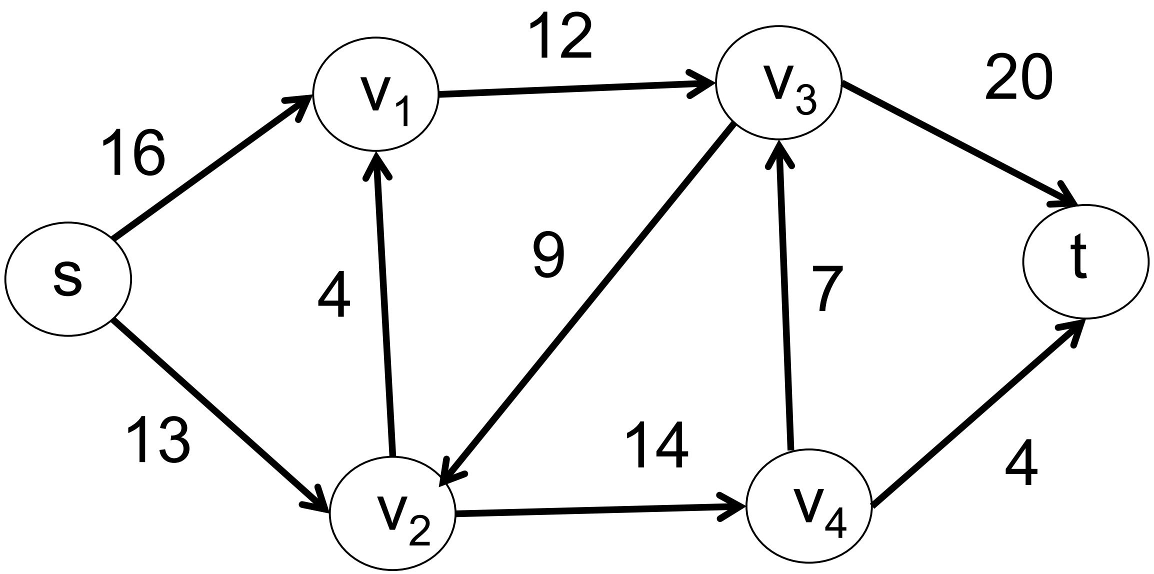
For each vertex a, the for loop executes |a.adjacent()| times.

$$\sum_{a \in V} |a.adjacent()| = |E|$$

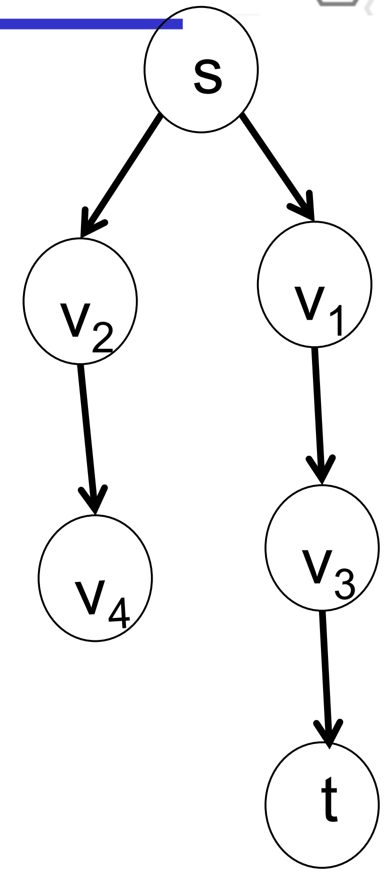
In total, O(|E|+|V|)***=O(|E|)***

Due to a connected graph.

Example

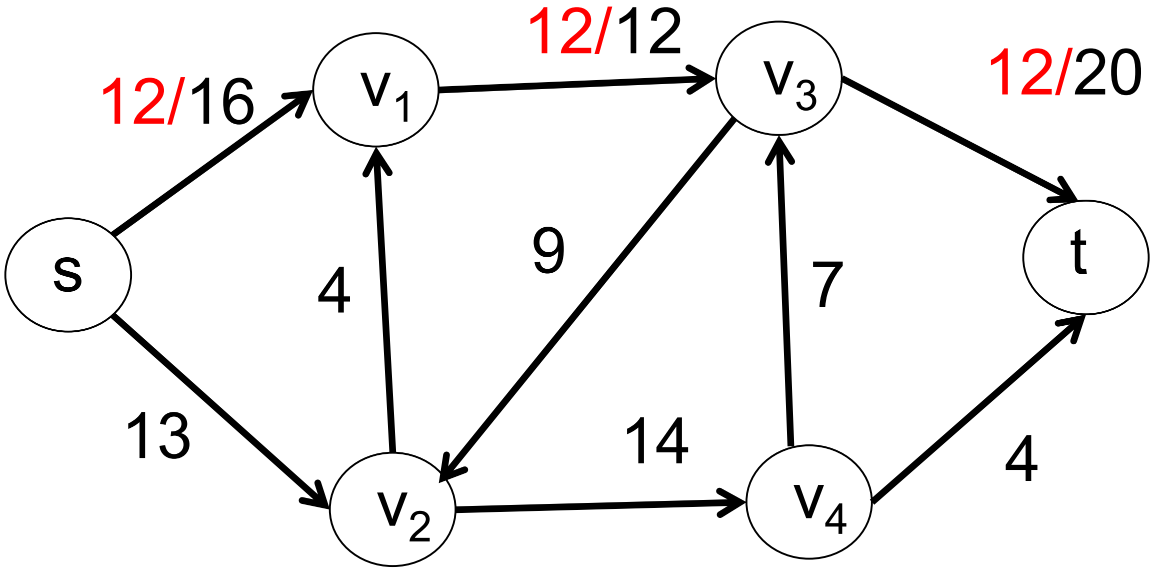


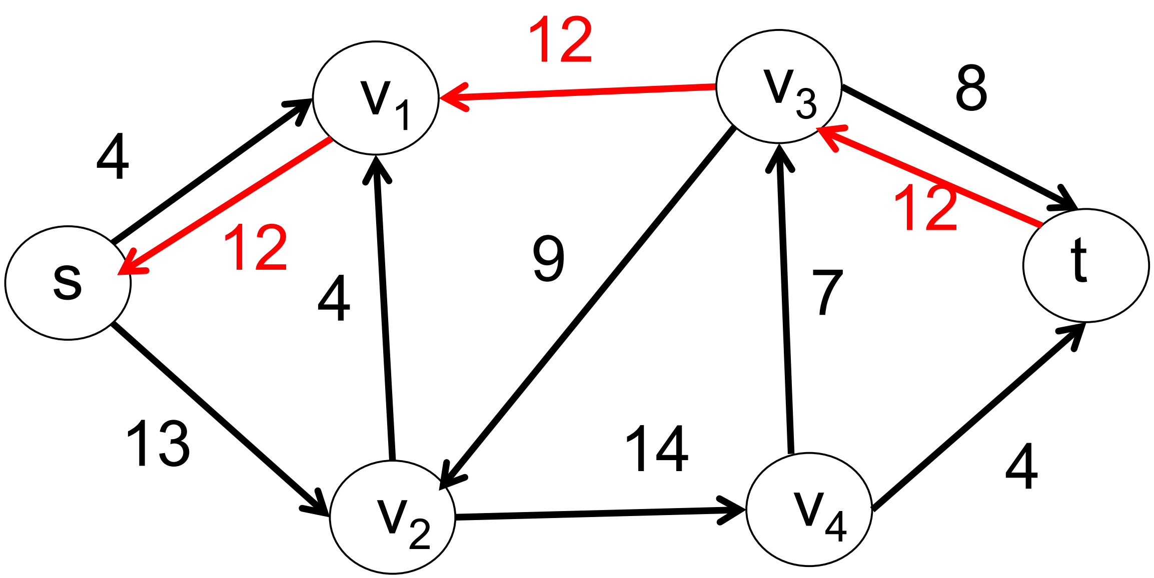
The original flow network and residual network

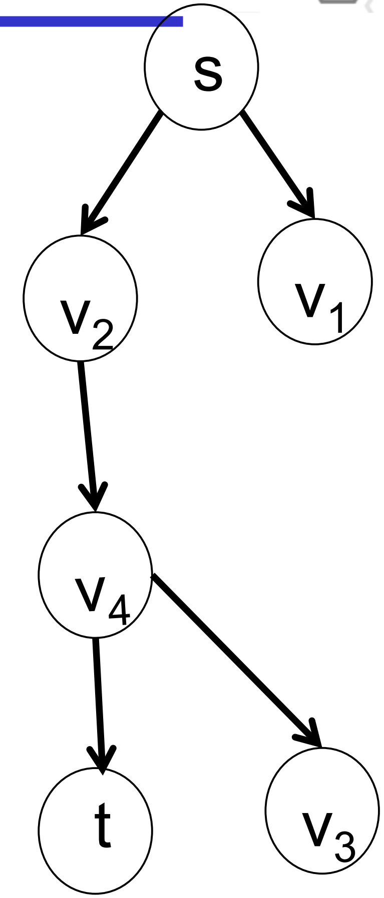


Shortest path:
p=<s, v1, v3, t>
$C_f(p)$=12

Shortest path: p=<s, v1, v3, t>, $C_f(p)$=12

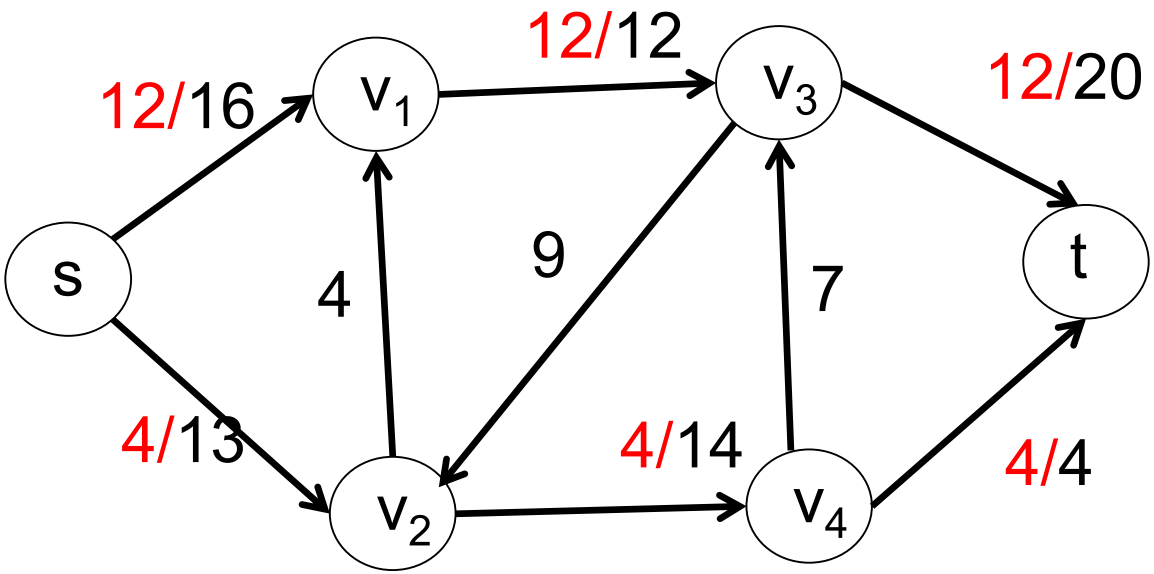


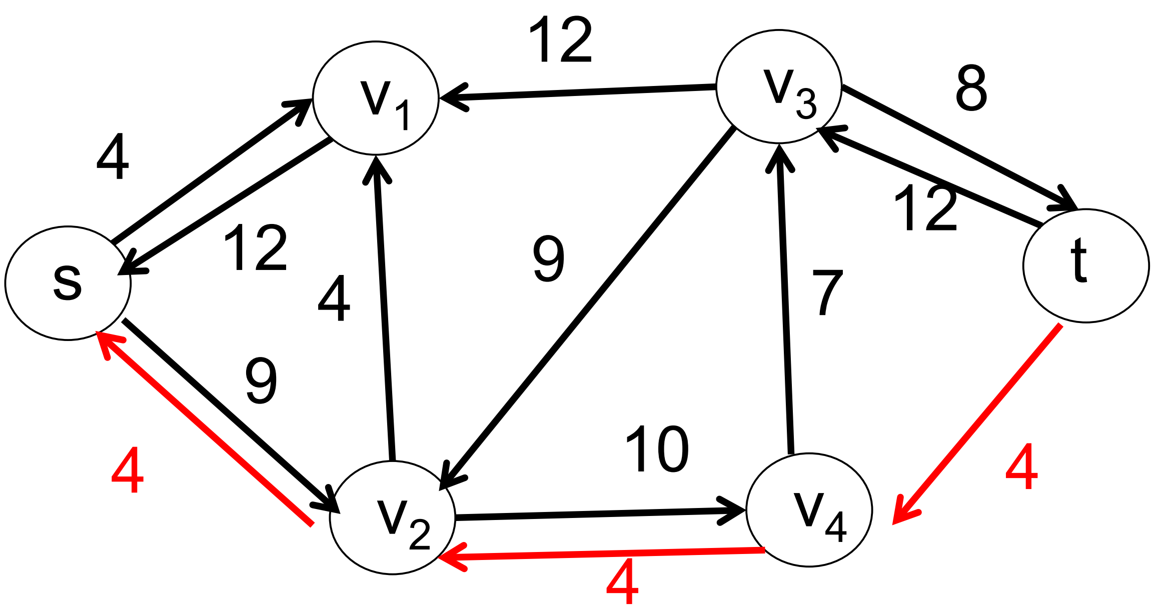


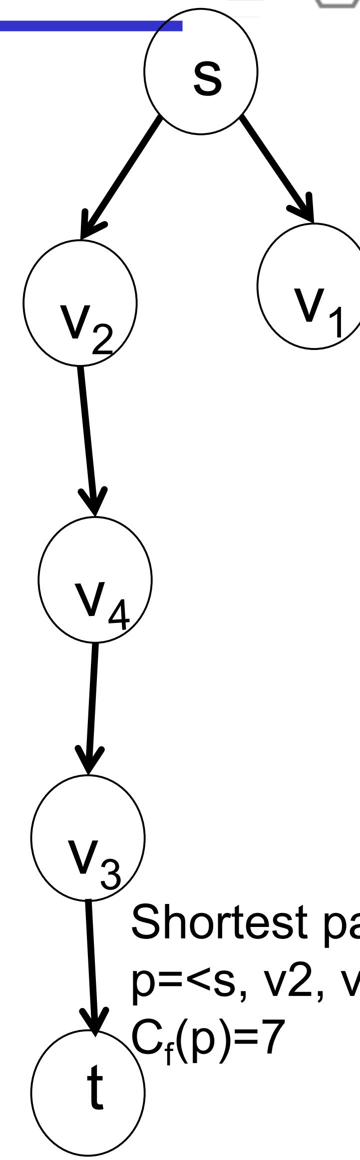


Shortest path:
p=<s, v2, v4, t>
$C_f(p)$=4

Shortest path: p=< s, v2, v4, t >, $C_f(p)=4$

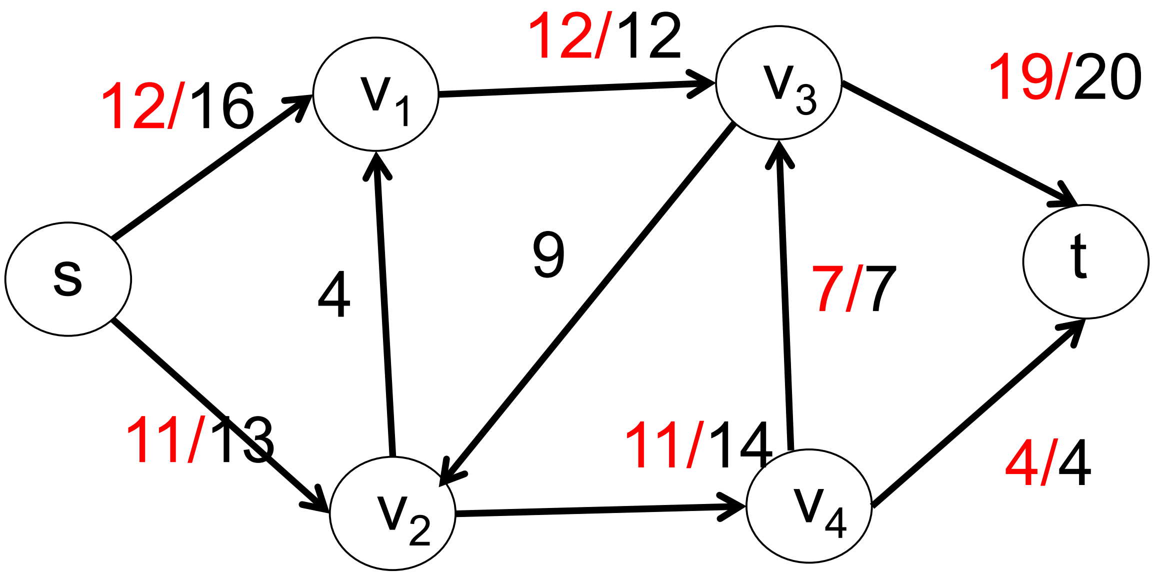


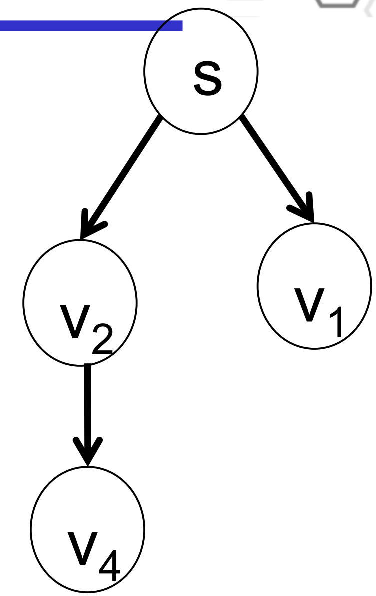




Shortest path:
p=<s, v2, v4, v3, t>
$C_f(p)=7$

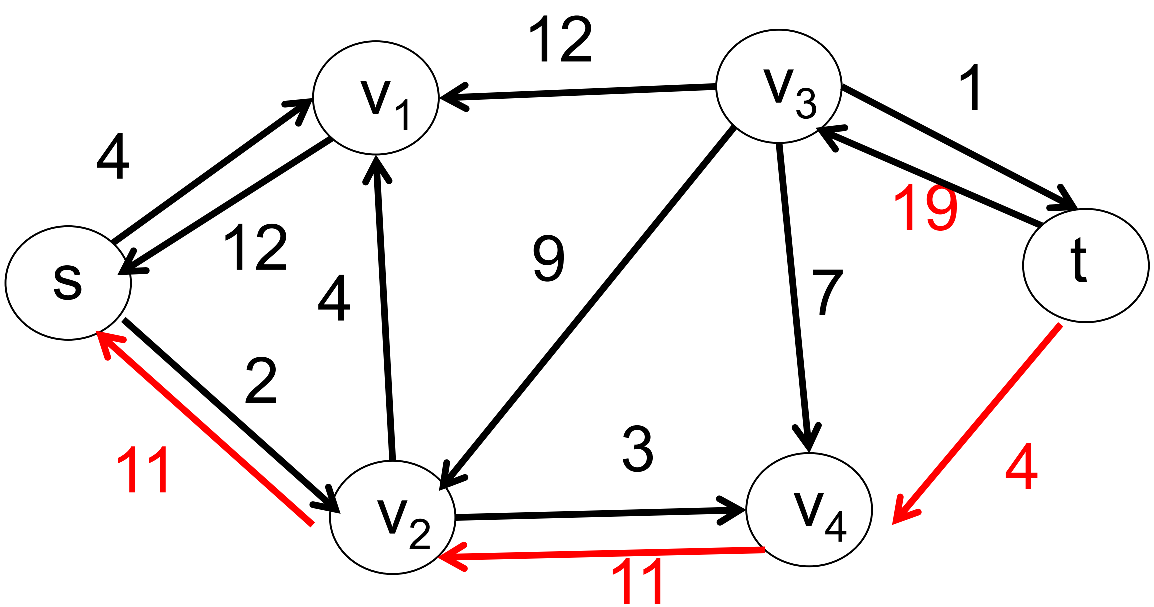
Shortest path: p=< s, v2, v4, v3, t >, $C_f(p)$=7





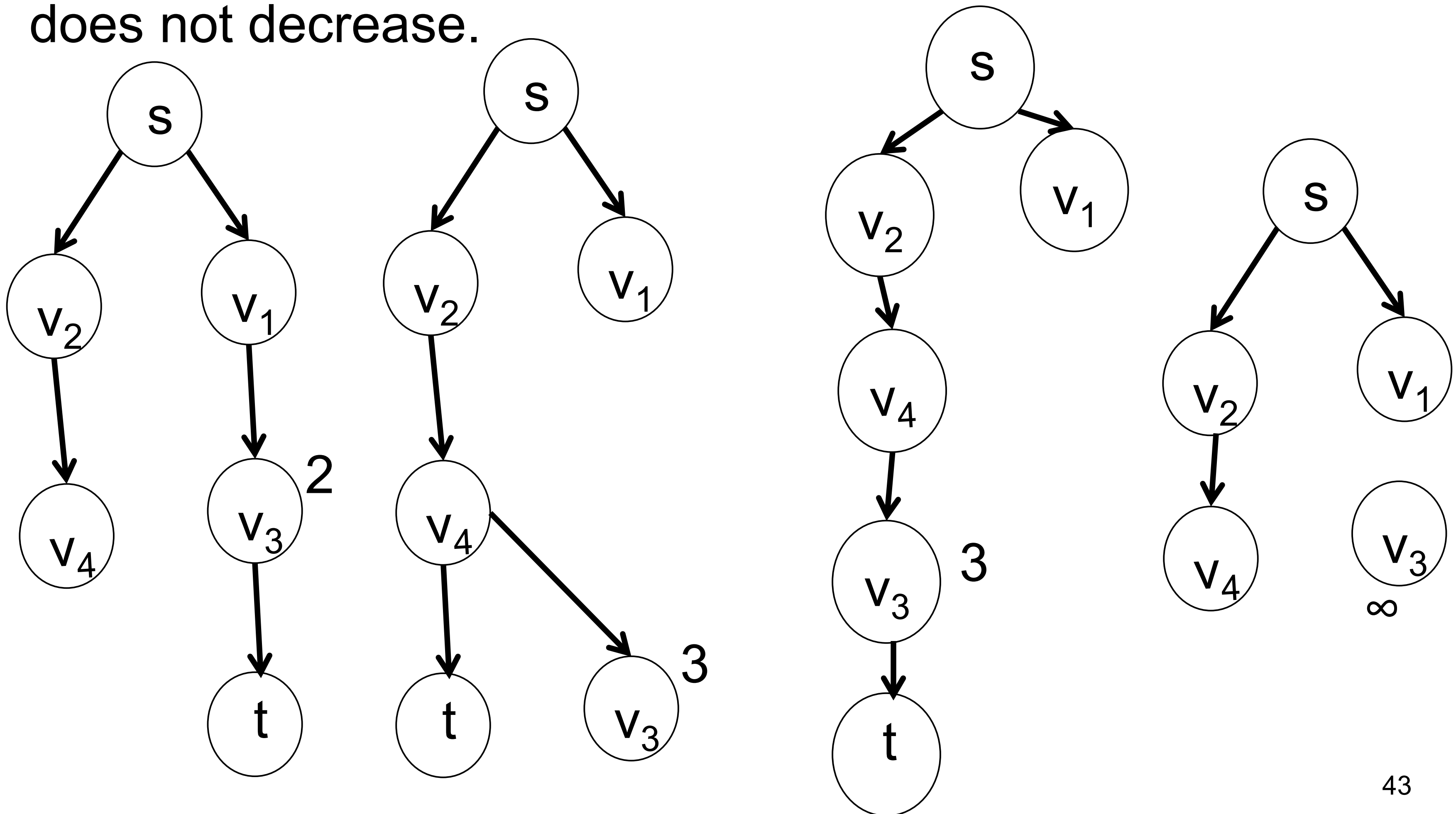
No path is able to connect s and t anymore.

Maximum-flow: 12+11=23



Non-decreasing shortest paths

- Consider a vertex v that is not the source and the sink, i.e., where $v \in V-\{s, t\}$.
- The shortest-path distance $\delta_f(s, v)$ in the residual network does not decrease.



Non-decreasing shortest paths

- Why $\delta_f(s, v)$ never decreases?
  - For a new residual network, we may add or delete edges from the previous residual network.
  - Deleting edges only increases the length of the shortest path $\delta_f(s, v)$.
  - Adding edges may decrease the length of the shortest path $\delta_f(s, v)$.
    - Only when adding "shortcuts"
    - The edges added in a residual network are opposite to the direction of the shortest path, so they are never "shortcuts".
  - Formal proof can be found in CLRS, Lemma 26.7, p 727.

Running time of Edmonds-Karp

- Each augmentation is $O(|E|)$
  - BFS

- How many augmentations in total can we have?
  - Each augmenting path has at least one critical edge.
  - Each of the $|E|$ edges can become critical at most $|V|/2$ times.
    - P 729, CLRS ***Theorem 26.8***
  - Thus, in total $O(|E||V|)$ times of augmentations.

- Thus, in total $O(|V||E|^2)$

Running time of Edmonds-Karp (2)

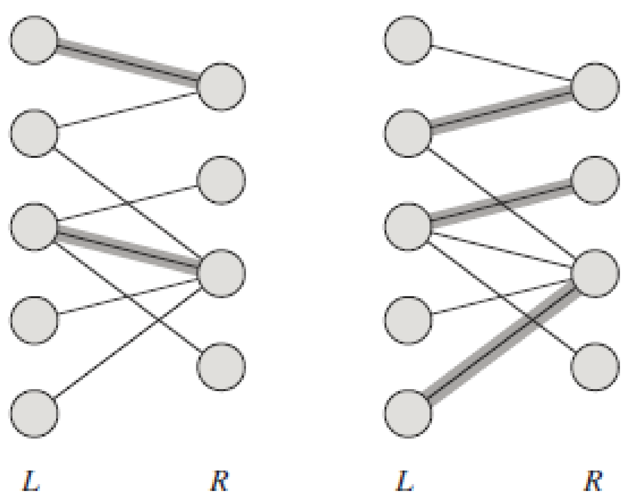
- An edge can be a critical edge at most |V|/2 times
  - Consider an edge (u, v) in a residual network G_f.
  - And assume that (u, v) is the critical edge on an augmenting path.
    - We have $\delta_f(s, v) = \delta_f(s, u) + 1$
  - After the augmentation, (u, v) disappears from the current residual network G_f.
  - (u, v) may reappear in a new residual network again after (v, u) is on an augmenting path in $G_{f'}$
    - We have $\delta_{f'}(s, u) = \delta_{f'}(s, v) + 1$
  - Due to the non-decreasing shortest path property we just saw
    - $\delta_f(s, v) \leq \delta_{f'}(s, v)$
    - $\delta_{f'}(s, u) = \delta_{f'}(s, v) + 1$
    - $\geq \delta_f(s, v) + 1$
    - $= \delta_f(s, u) + 2$
    - The distance from source s to u increases by at least 2.
  - The longest possible distance from s to u is |V|-2
    - An edge can be a critical edge for at most (|V|-2)/2 times.

Agenda

- Flow networks, flows, maximum-flow problem
- The Ford-Fulkerson method
- The Edmonds-Karp algorithm
- Maximum-bipartite-matching
- Spatial crowd sourcing

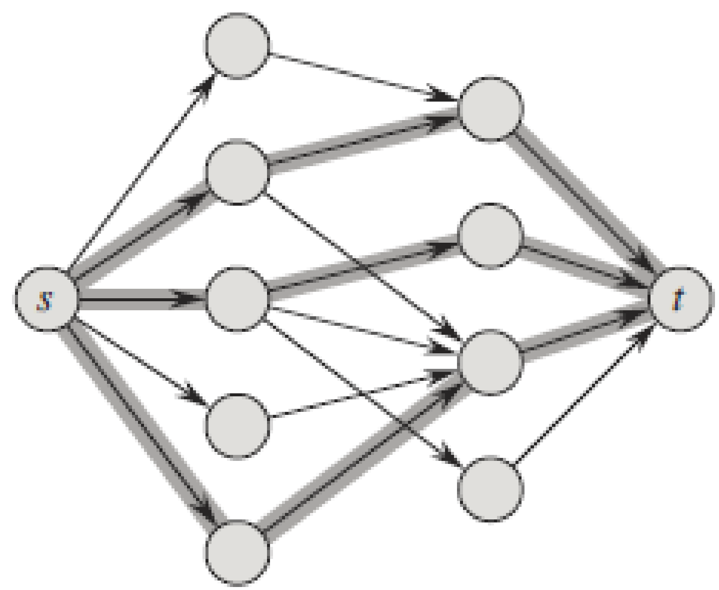
Maximum-bipartite-matching

- A ***bipartite graph*** is an undirected graph G=(V, E)
  - Vertex set V can be partitioned into L and R, where L and R are disjoint and $V = L \cup R$.
  - All edges in E go between L and R. For each $(u, v) \in E$, we have $u \in L$ and $v \in R$ or $u \in R$ and $v \in L$.
- Given an undirected graph G=(V, E), a ***matching*** is a subset of edges $M \subseteq E$ such that for each vertex $v \in V$, at most one edge of M is incident on v.
- ***Maximum matching*** is a matching of maximum cardinality.



Finding a maximum bipartite matching

- Create a source vertex s and a sink vertex t.
- Create an edge from s to every vertex in L.
- Create an edge from every vertex in R to t.
- Assign each edge with capacity 1.
- Identify the maximum flow.
- Those edges from L to R whose flow is 1 constitutes the maximum matching.

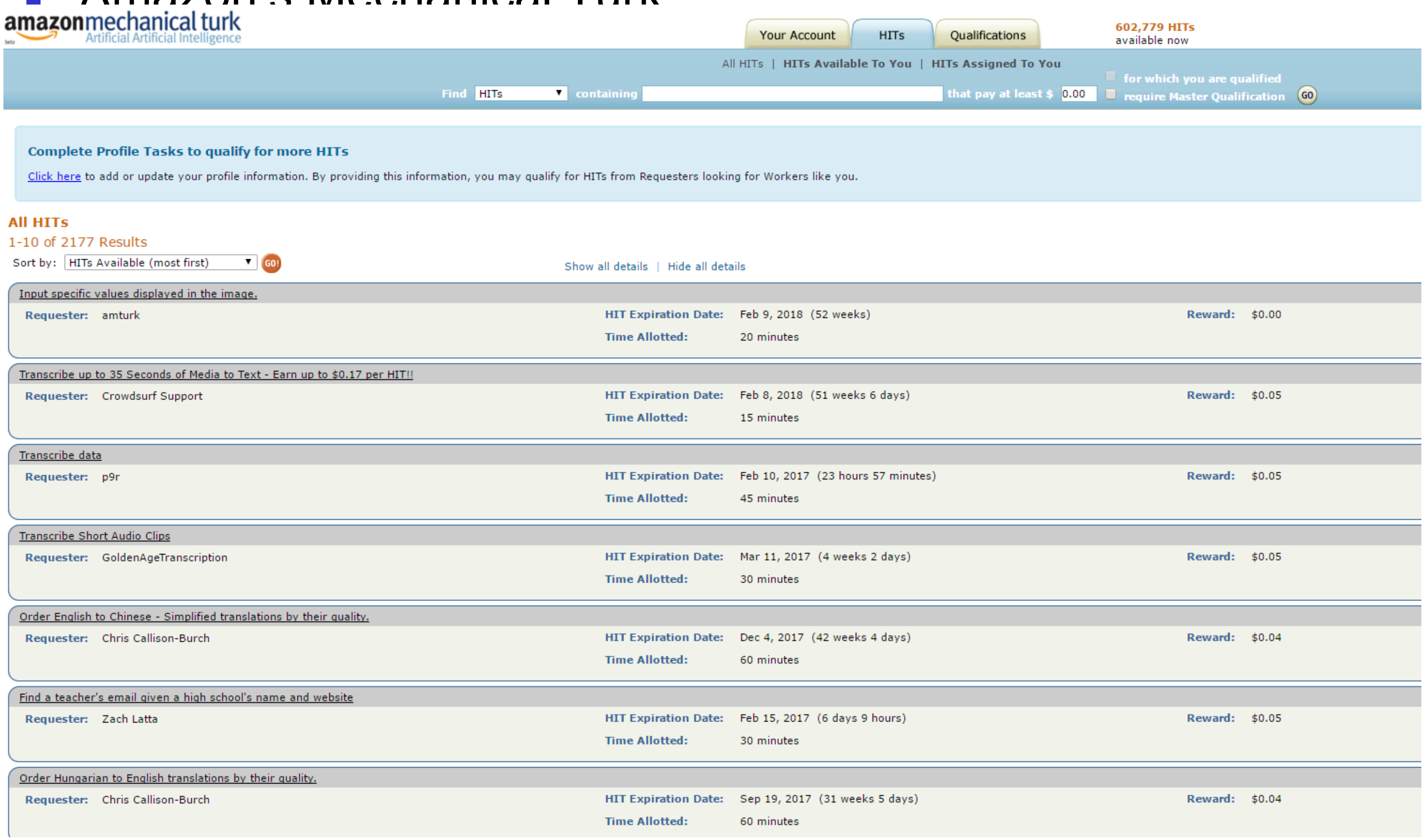


Agenda

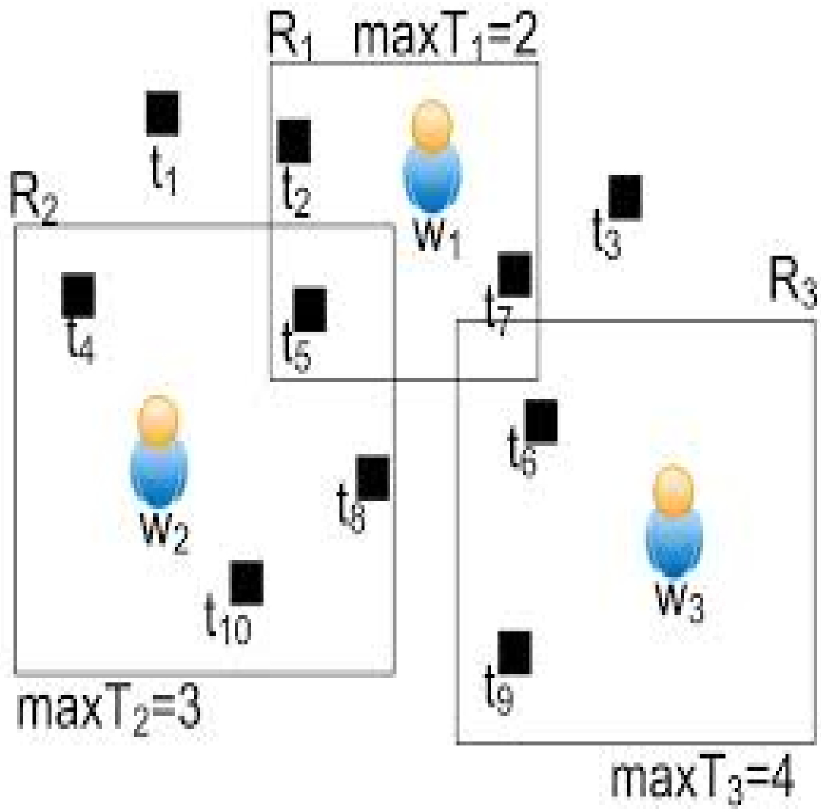
- Flow networks, flows, maximum-flow problem
- The Ford-Fulkerson method
- The Edmonds-Karp algorithm
- Maximum-bipartite-matching
- Spatial crowd sourcing

Spatial Crowdsourcing

- Crowdsourcing
  - Tasks and workers
  - Amazon’s Mechanical Turk
-



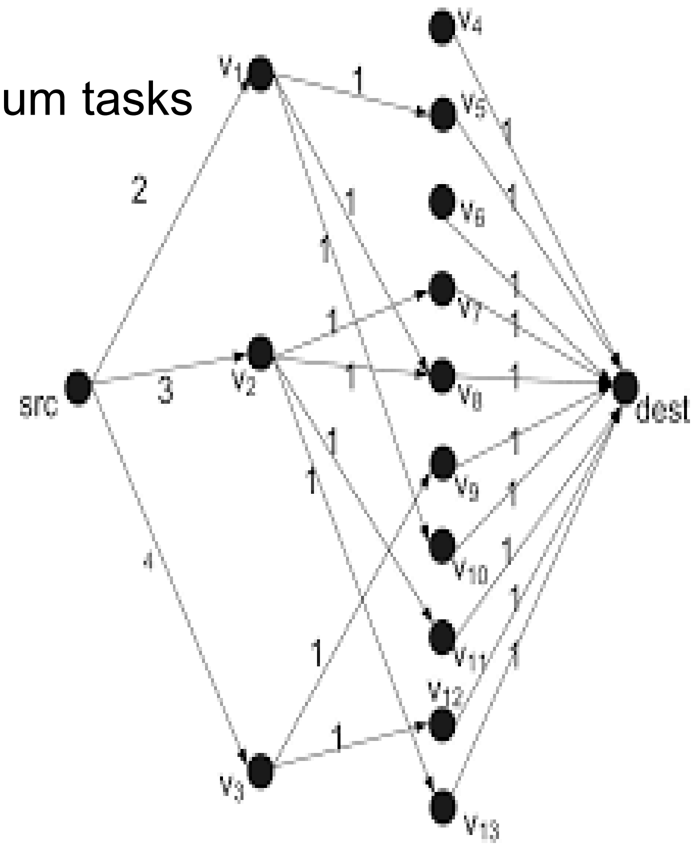
Maximum Task Assignment Problem



- Workers W ={w_1, w_2, w_3}
- Tasks T = {t_1,t_2,t_3,t_4,t_5,t_6,t_7,t_8,t_9,t_{10}}
- Assignment instance s_i=<w,t>
- Worker has constraints to satisfy:
  - Spatial Range R_i
  - Maximum tasks $maxT_i$

Reducing to Maximum Flow Problem

Maximum tasks $maxT_i$



- Flow network graph G=(V,E), where:
  - V contains $|w_i|+|t_i|+2$ vertices
  - E contains $|w_i|+|t_i|+m$ edges
- Edges between workers and tasks are added if the tasks lie in the spatial regions of workers
- Every task can be assigned to only one worker.

ILO of Lecture 3

- Flow network
  - to understand the formalization of flow networks and flows; and the definition of the maximum-flow problem.
  - to understand the Ford-Fulkerson method for finding maximum flows.
  - to understand the Edmonds-Karp algorithm and to be able to analyze its worst-case running time;
  - to be able to apply the Ford and Fulkerson method to solve the maximum-bipartite-matching problem.

Lecture 4

- Greedy Algorithms
  - to understand the principles of the greedy algorithm design technique;
  - to understand the greedy algorithms for activity selection and Huffman coding, to be able to prove that these algorithms find optimal solutions;
  - to be able to apply the greedy algorithm design technique.