Prepared by: [Arun Konagurthu]

FIT3155 S1/2020: Advanced Data Structures and Algorithms

Week 11 Lecture:

Network Flow and Assignment problems

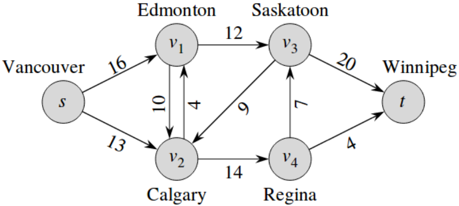
Faculty of Information Technology, Monash University

Source material and recommended reading

- Cormen et al., Introduction to Algorithms. Chapter 26.

Flow networks – Introduction

- **Networks** are **Directed graphs**
- A **flow network** is a connected **directed graph** where
  - there is (often) a single source vertex and a single sink/destination vertex;
  - each edge has a stated (non-negative) **capacity**...
  - ...giving the **maximum amount** of **flow** that edge can carry;



Flow networks – Introduction

Flow networks model many real-world problems

- Traffic flowing on roads.
- Electric current flowing through electrical circuits.
- Water flowing through an assembly of pipes.
- Information flowing through communication networks
- Can be applied to many scenarios (unrelated to physical flows).

Flow

- Flow is an **assignment** of how much material ('stuff') can flow through each edge in the flow network given its stated edge capacity.

Key property: flow conservation

In a flow network, the amount flowing **into** any vertex (through all of its incoming edges)...

IS STRICTLY EQUAL TO

...the amount flowing **out** of that vertex (through all of its outgoing edges).

We will clarify the precise details in the upcoming slides.

Some basic notations to consider in this lecture

Flow network: denoted as $G(V, E, C)$, where

- V = set of vertices
- E = set of directed edges
- C = set of capacities (corresp. to the set of edges, E)

Source vertex is denoted as s (has no incoming edges)

Sink/Destination vertex is denoted as t (has no outgoing edges)

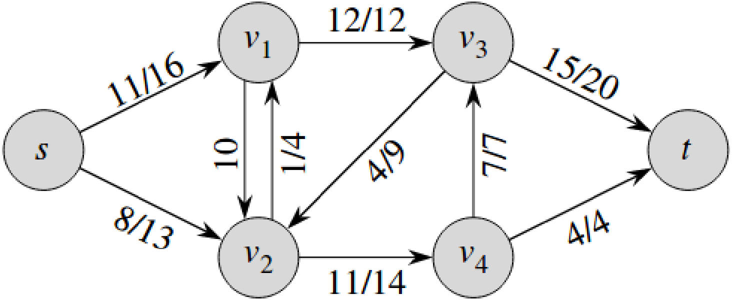
Set of incoming edges into a vertex $v \in V$ is denoted as $E_{in}(v)$.

Set of outgoing edges from a vertex $v \in V$ is denoted as $E_{out}(v)$.

Capacity of any edge (i.e., the maximum amount of flow an edge can carry) $e \in E$ is denoted as **cap**(e).

Property 1 of a flow network: Capacity constraint

Capacity constraint For each edge $e \in E$, its **flow**, denoted as $\texttt{flow}(e)$, **is bounded by the capacity** of its edge: $0 \leq \texttt{flow}(e) \leq \texttt{cap}(e)$.



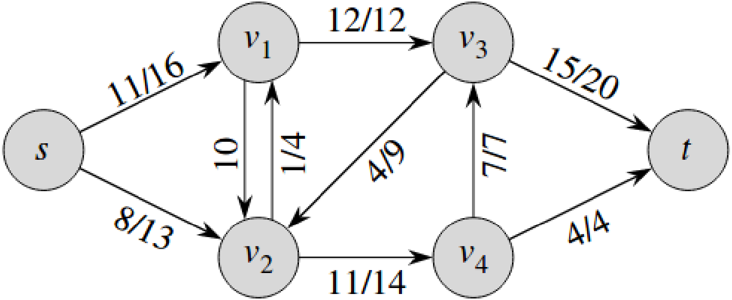
In the example above, (most) edges are labeled with two numbers, separated by a slash ('/'). The first number is the assigned flow and the second number is the edge's given capacity. Edges with only one number implies that their flow is 0, and only the capacity is shown.

Property 2 of a flow network: Flow conservation

Flow conservation

For any vertex v (that is not either **source** or **sink**), the amount of flow into that vertex from its in-coming edges is same as the amount of flow going out from its out-going edges – Formally:

$$\sum_{\forall e_{in} \in E_{in}(v)} \mathbf{flow}(e_{in}) = \sum_{\forall e_{out} \in E_{out}(v)} \mathbf{flow}(e_{out}).$$

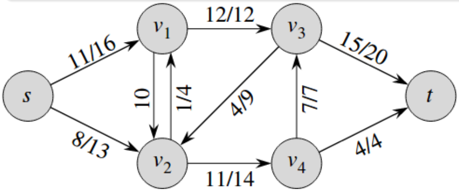


The value of a flow within the whole network

Value of a flow

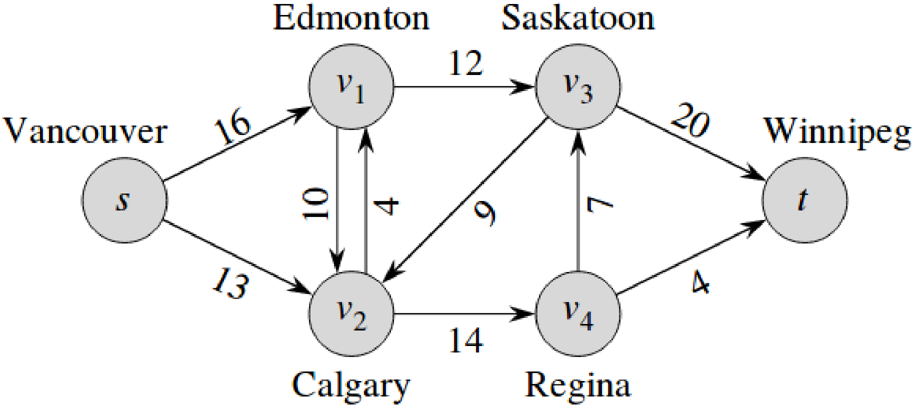
If a flow network satisfies the two properties stated on Slides #8 and #9, its **value** is the **total flow out of the source vertex**. Equivalently, this should be same the **total flow into sink vertex**.

$$\text{Flow value} = \sum_{\forall \langle s,x\rangle \in E_{out}(s)} \texttt{flow}(\langle s, x\rangle) = \sum_{\forall \langle y,t\rangle \in E_{in}(t)} \texttt{flow}(\langle y, t\rangle).$$



In the network above, the flow **value** is 19. This is derived by summing up the flows on the outgoing edges (11+8=19) at the source vertex s. Equivalently, this is same as summing up the flows on the incoming edges (15+4=19) at the sink vertex t.

Maximum-flow problem



Problem statement: Given some **flow network**, what is the **maximum value** of the flow that can be sent from source (s) to sink (t) **without** violating the **flow conservation** on each vertex in the network?

Ford and Fulkerson proposed an influential approach to solving the maximum-flow problem

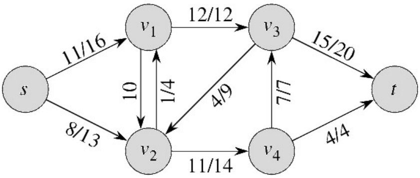
Introduction to Ford-Fulkerson's algorithm

- A method to solve the maximum-flow problem proposed in 1956 by two mathematician, Lester Ford and Delbert Fulkerson: [Link to original paper]
- This method proposed three cute and v. v. influential ideas now common in many optimization problems:
  1. **Residual** flow networks
  2. **Augmenting** paths
  3. **Cuts**

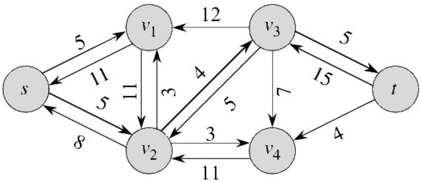
What is the **residual network** for a given flow network?

Given a flow network G:

- The residual network $G_{residual}$ has the **same number of vertices** as the original network G.
- However, for every directed edge $\langle u, v \rangle$ in the original network G:
  - there is a directed edge $\langle u, v \rangle$ (in the same direction) in $G_{residual}$ whose capacity is $c(\langle u, v \rangle) - f(\langle u, v \rangle)$, if this quantity is greater than 0.
  - there is a directed edge $\langle v, u \rangle$ (in the reverse direction) in $G_{residual}$ whose capacity is $f(\langle u, v \rangle)$, if this quantity is greater than 0.



original network G

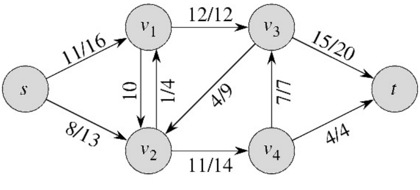


residual network $G_{residual}$ of G

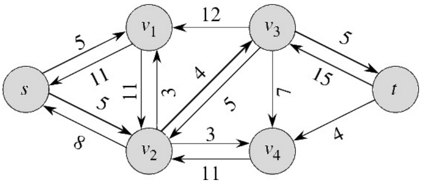
What is the residual network for a given flow network?

Given a flow network G:

- The residual network $G_{residual}$ has the **same number of vertices** as the original network G.
- However, for every directed edge $\langle u, v\rangle$ in the original network G:
  - there is a directed edge $\langle u, v\rangle$ (in the **same** direction) in $G_{residual}$ whose capacity is $c(\langle u, v\rangle) - f(\langle u, v\rangle)$, if this quantity is greater than 0.
  - there is a directed edge $\langle v, u\rangle$ (in the **reverse** direction) in $G_{residual}$ whose capacity is $f(\langle u, v\rangle)$, if this quantity is greater than 0.



original network G

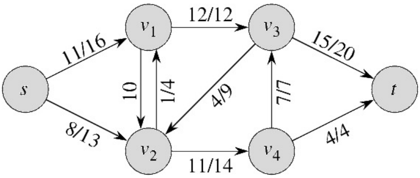


residual network $G_{residual}$ of G

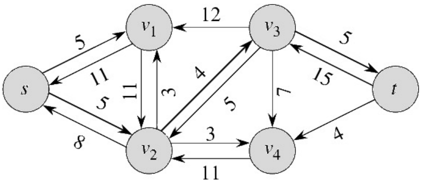
What is the **residual network** for a given flow network?

Given a flow network G:

- The residual network $G_{residual}$ has the **same number of vertices** as the original network G.
- However, for every directed edge $\langle u, v\rangle$ in the original network G:
  - there is a directed edge $\langle u, v\rangle$ (in the **same** direction) in $G_{residual}$ whose capacity is $c(\langle u, v\rangle) - f(\langle u, v\rangle)$, if this quantity is greater than 0.
  - there is a directed edge $\langle v, u\rangle$ (in the **reverse** direction) in $G_{residual}$ whose capacity is $f(\langle u, v\rangle)$, if this quantity is greater than 0.



original network G

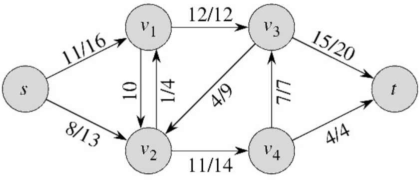


residual network $G_{residual}$ of G

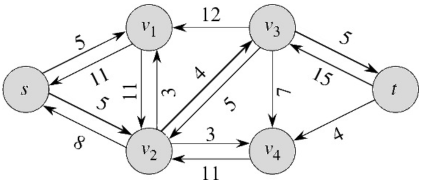
What is the **residual network** for a given flow network?

Given a flow network G:

- The residual network $G_{residual}$ has the **same number of vertices** as the original network G.
- However, for every directed edge $\langle u, v \rangle$ in the original network G:
  - there is a directed edge $\langle u, v \rangle$ (in the **same** direction) in $G_{residual}$ whose capacity is $c(\langle u, v \rangle) - f(\langle u, v \rangle)$, if this quantity is greater than 0.
  - there is a directed edge $\langle v, u \rangle$ (in the **reverse** direction) in $G_{residual}$ whose capacity is $f(\langle u, v \rangle)$, if this quantity is greater than 0.



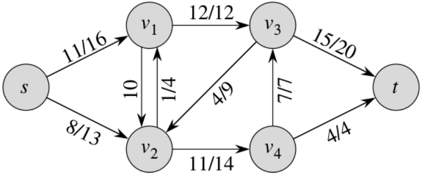
original network G



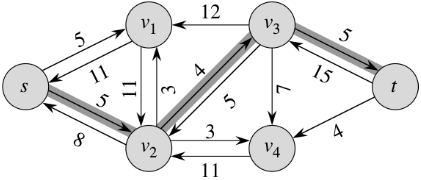
residual network $G_{residual}$ of G

Augmenting path in the residual network

Any **simple path** (i.e. path without repeating vertices) from source s to sink t in the residual network of G is an **augmenting path**.



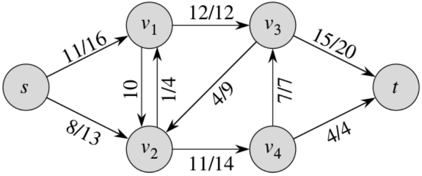
original network G



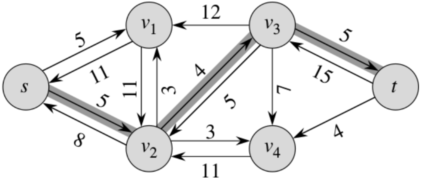
augmenting path in $G_{residual}$ of G

Augmenting path in the residual network

Any **simple path** (i.e. path without repeating vertices) from source s to sink t in the residual network of G is an **augmenting path**.



original network G



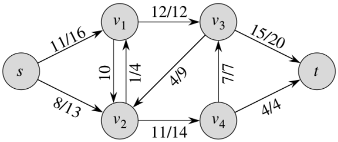
augmenting path in $G_{residual}$ of G

What is an augmenting path useful for?

The flow **value** of G can be augmented by the **minimum** capacity (or **flow bottleneck**) observed on the edges along the augmenting path in $G_{residual}$. In the example above, the **bottleneck** along the augmenting path (shaded edges) is 4. So, the flow in the original graph can be augmented by 4 (see how on next slide).

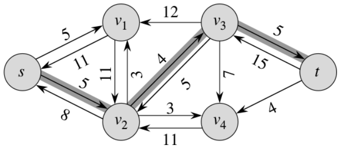
Augmenting flow value in G using any augmenting path in $G_{residual}$

Flow network G



flow value=19

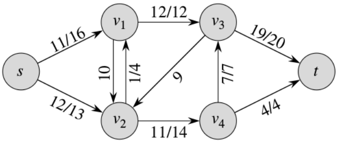
Augmenting path in $G_{residual}$



flow bottleneck = 4

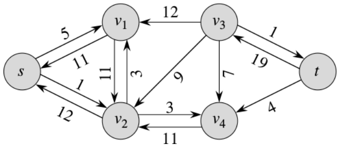
Augmented flow network:

Augmented flow network



flow value=23

Corresp. residual network



(details explained in the lecture)

Ford-Fulkerson – rough sketch

```
Ford-Fulkerson(Network G, Source s, Sink t) {
    initialize the flow on each edge in G to 0.

    G_residual = G
    while (there exists a path p from s to t in G_residual) {
        augment G based on bottleneck of path p
        update G_residual
    }
    return flow;
}
```

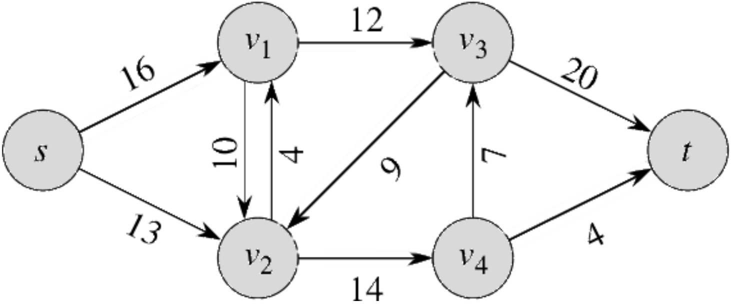
Ford-Fulkerson's algorithm – pseudocode

```
/* Input: (1) flow network G(V,E,C).
          (2) s in V is the source
          (3) t in V is the sink */
function Ford-Fulkerson(G, s, t) {
    flow[1..|E|] = 0;  // init flow on each edge in G to 0.
    G_residual = G; // init residual network to G. Capacity of edges
                    // ...in G_residual = assigned flow of edges in G

    while (there exists a path p from s to t in G_residual) {
       // Let path p contain edges {e_1, e_2, ...}
       bottleneck = minimum_i (capacity[e_i])/*forall e_i in p*/

       for (each edge e_i=<u,v> in p) {                        // A
          if (<u,v> in G) {                                    // U
              flow[<u,v>] = flow[<u,v>] + bottleneck;          // G
          }                                                    // M
          else {                                               // E
              flow[<v,u>] = flow[<v,u>] - bottleneck;          // N
          }                                                    // T
       }
       Compute G_residual of G using current flow assignments.
    }
    return flow;
}
```

Ford-Fulkerson's algorithm – Example (Initialization step)

Input network G whose flow **value** we want to maximize is below. Init all **flow**$(e) = 0$ (only **cap**(e) values are shown below, since all **flow**$(e) = 0$).

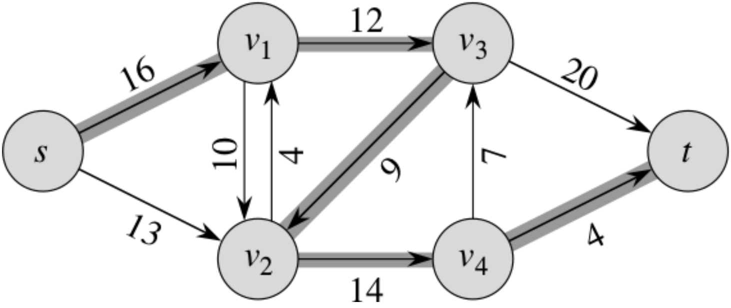
Current flow **value** of $G = 0$.



Ford-Fulkerson's algorithm – Example...cont'd (Iteration-1)

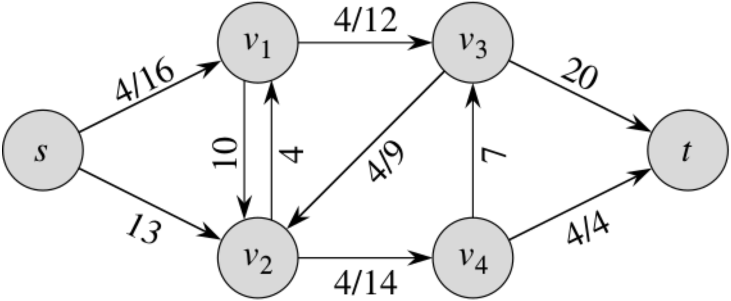
When all edge's **flow**$(e) = 0$, then $G_{residual}$ is equivalent to G. Find a simple path p from s to t.

Flow **bottleneck** of p is minimum over all edge capacities along the path $(= \min\{16, 12, 9, 14, 4\} = 4)$.



Ford-Fulkerson's algorithm – Example...cont'd (Iteration-1)

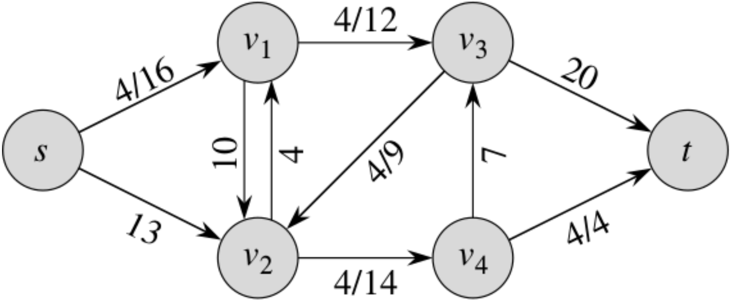
Update the edges along the path in the original network by the permissible flow bottleneck ($= 4$)



Ford-Fulkerson's algorithm – Example...cont'd (Iteration-1)

Update the edges along the path in the original network by the permissible flow bottleneck ($= 4$)

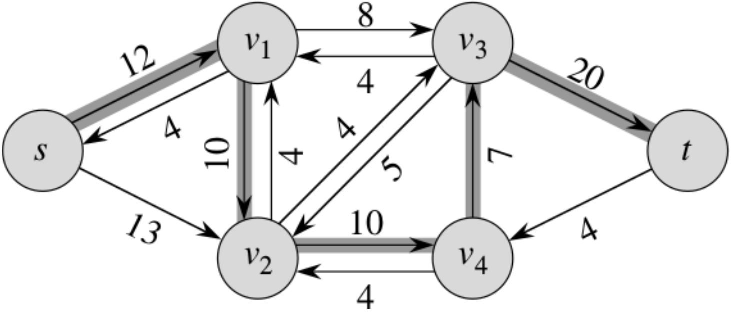
(Augmented) Current flow **value** of $G = 4$.



Ford-Fulkerson's algorithm – Example...cont'd (Iteration-2)

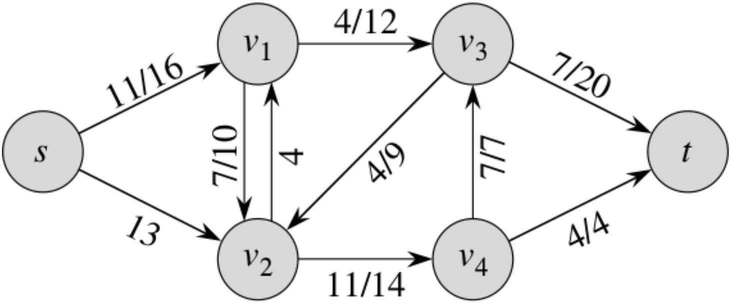
Build residual network and find any path p from s to t.

Flow **bottleneck** of p is minimum over all edge capacities along the path $(= \min\{12, 10, 10, 7, 20\} = 7)$.



Ford-Fulkerson's algorithm – Example...cont'd (Iteration-2)

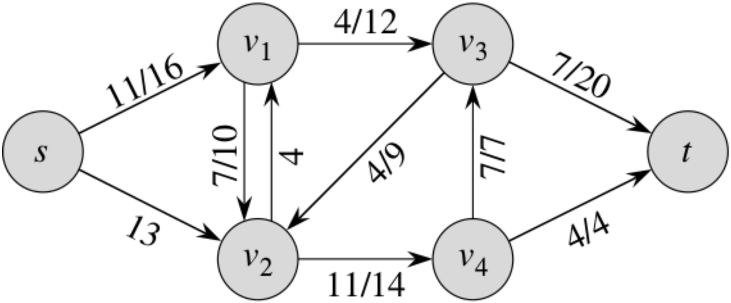
Update the edges along the path in the original network by the permissible flow bottleneck ($= 7$)



Ford-Fulkerson's algorithm – Example...cont'd (Iteration-2)

Update the edges along the path in the original network by the permissible flow bottleneck ($= 7$)

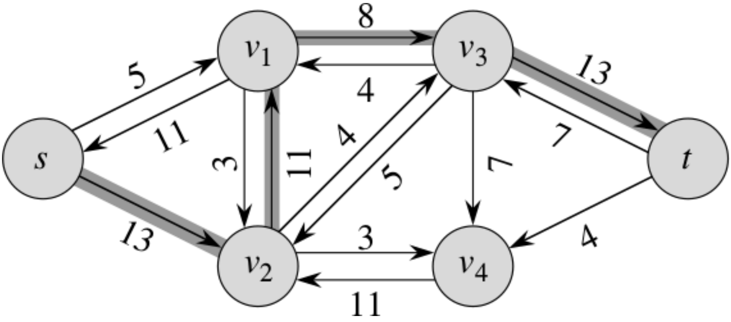
(Augmented) Current flow **value** of $G = 11$.



Ford-Fulkerson's algorithm – Example...cont'd (Iteration-3)

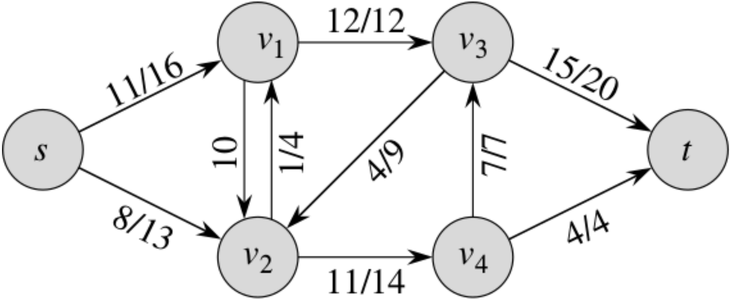
Build residual network and find any path p from s to t.

Flow **bottleneck** of p is minimum over all edge capacities along the path $(= \min\{13, 11, 8, 13\} = 8)$.



Ford-Fulkerson's algorithm – Example...cont'd (Iteration-3)

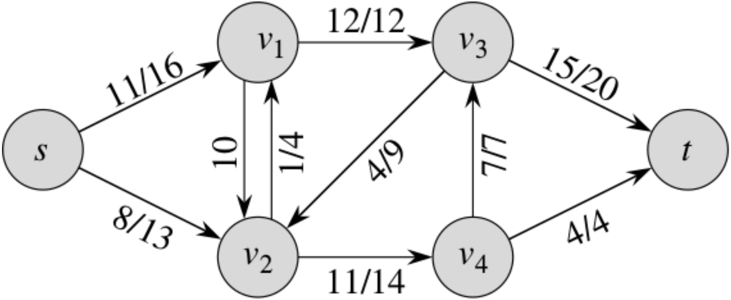
Update the edges along the path in the original network by the permissible flow bottleneck $(= 8)$



Ford-Fulkerson's algorithm – Example...cont'd (Iteration-3)

Update the edges along the path in the original network by the permissible flow bottleneck ($= 8$)

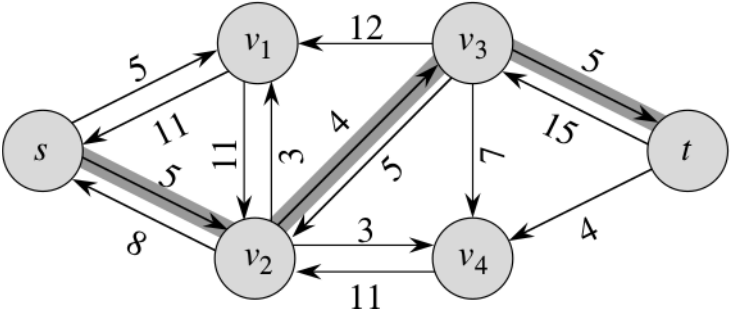
(Augmented) Current flow **value** of $G = 19$.



Ford-Fulkerson's algorithm – Example...cont'd (Iteration-4)

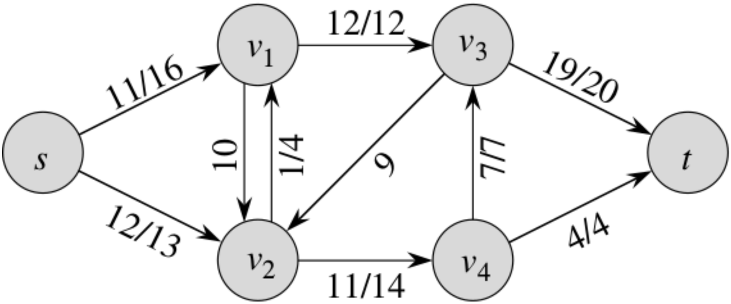
Build residual network and find any path p from s to t.

Flow **bottleneck** of p is minimum over all edge capacities along the path $(= \min\{5, 4, 5\} = 4)$.



Ford-Fulkerson's algorithm – Example...cont'd (Iteration-4)

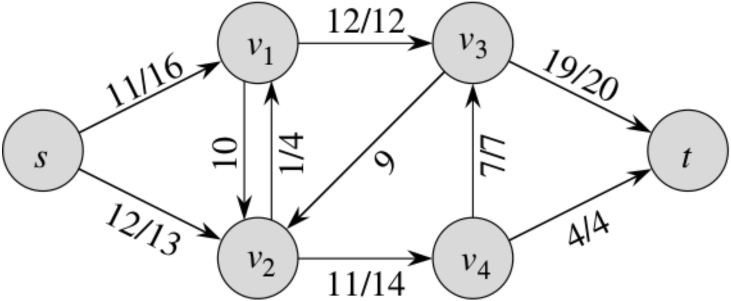
Update the edges along the path in the original network by the permissible flow bottleneck ($= 4$)



Ford-Fulkerson's algorithm – Example...cont'd (Iteration-4)

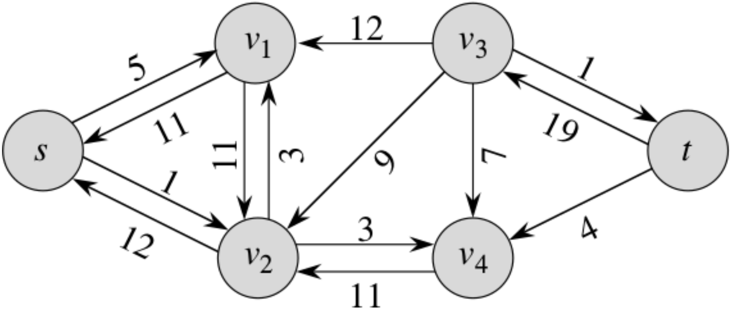
Update the edges along the path in the original network by the permissible flow bottleneck ($= 4$)

(Augmented) Current flow **value** of $G = 23$.



Ford-Fulkerson's algorithm – Example...cont'd (Iteration-5)

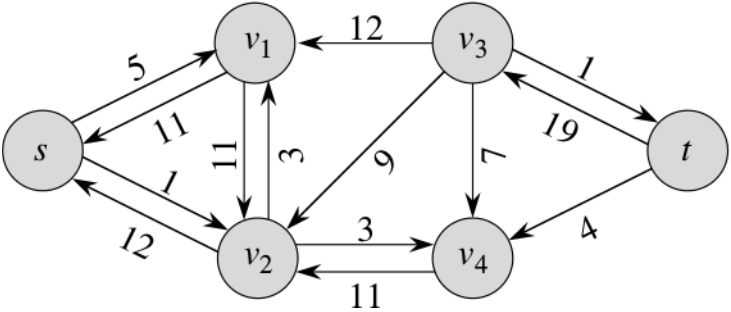
Build residual network and find any path p from s to t. (Try and find one!)



Ford-Fulkerson's algorithm – Example...cont'd (Iteration-5)

Build residual network and find any path p from s to t. (Try and find one!)

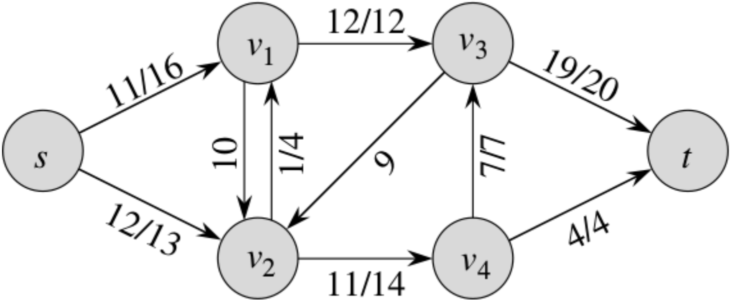
No path from s to t. STOP!!!



Ford-Fulkerson's algorithm – Example...cont'd

The state of flow assignments when the program terminated, gives you the maximum flow **value** in the given network.

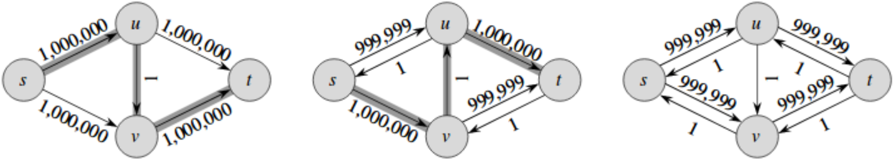
The **maximum** flow **value** of $G = 23$.



Analysis of Ford-Fulkerson Algorithm

- The running time of Ford-Fulkerson on a network $G(V, E, C)$ depends on how we find the augmenting path p and how many times the outer-while loop runs (see Line 9 on Slide #17).
- Of all path algorithms discussed in your previous study, a natural choice of finding a path in $G_{residual}$ is BFS (to find the shortest path from s to t by treating $G_{residual}$ edges as unweighted).
  - Number of vertices in $G_{residual} = |V|$
  - Number of edges in $G_{residual} \leq 2|E|$
  - Therefore, worst-case time to find a path in $G_{residual}$ is $O(|V| + 2|E|) = O(|E|)$
- For a flow network with integer capacities and the **maximum flow value** of $\texttt{Value}_{max}$, the outer while loop executes $\texttt{Value}_{max}$ times in the worst-case (Why???).
- Therefore, total run-time is $O(|E|\texttt{Value}_{max})$

An example flow network for which Ford-Fulkerson can take $O(|E|\mathtt{Value}_{max})$



Two iterations of Ford-fulkerson algorithm where the flow is augmented by 1 each iteration. In the worst case, there will be $2,000,000$ such iterations before the algorithm terminates.

In general, the total run-time is $O(|E|\mathtt{Value}_{max})$ which is a pseudo-polynomial-time algorithm.

Two strategies to implement of Ford-Fulkerson's general method

Ford-Fulkerson method really does not specify which augmenting path to use if there is more than one choice to be made.

Two implements that do NOT make arbitrary choices for augmenting paths are:

1. Dinic/Edmonds-Karp augmentation on $s \rightsquigarrow t$ path with **fewest edges**
2. Edmonds-Karp augmentation on $s \rightsquigarrow t$ path with **largest bottleneck**.

These two strategies guarantee to run in **polynomial** time.

Dinic/Edmonds-Karp augmentation on (s) $\rightsquigarrow$ (t) path with **fewest edges**

A straightforward approach to find (s) $\rightsquigarrow$ (t) augmenting path with **fewest edges** is:

- Run BFS from (s) to find (s) $\rightsquigarrow$ (t) path in $G_{residual}$.

Edmonds-Karp augmentation on (s) $\rightsquigarrow$ (t) path with **`largest bottleneck`**

A straightforward approach to find (s) $\rightsquigarrow$ (t) augmenting path with **`largest bottleneck`** is:

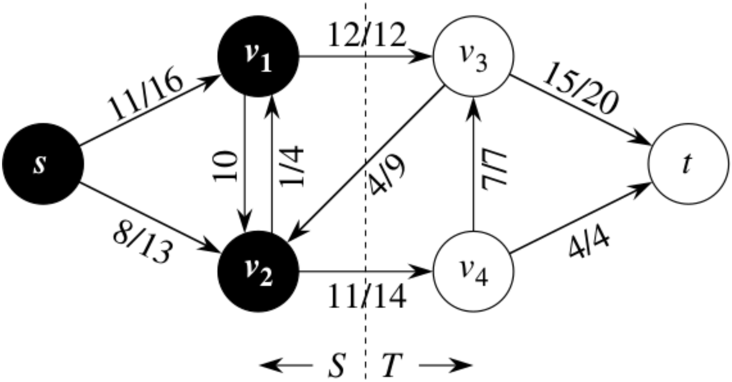
- Grow a spanning tree M starting from (s).
- In each iteration, choose the **`highest capacity`** across the cut defined by the vertices in M.
- Repeat the above until M grows to contain (t).

Correctness – How do we know Ford-Fulkerson algorithm terminates with **maximum flow value**?

Ford-Fulkerson in their influential paper proved what is called the **Minimal cut theorem**, which is variably called "**Min-cut Max-Flow**" (or sometimes "**Max-Flow Min-Cut**") theorem. But to understand this, we need to explore the notion of a **cut** in a flow network.

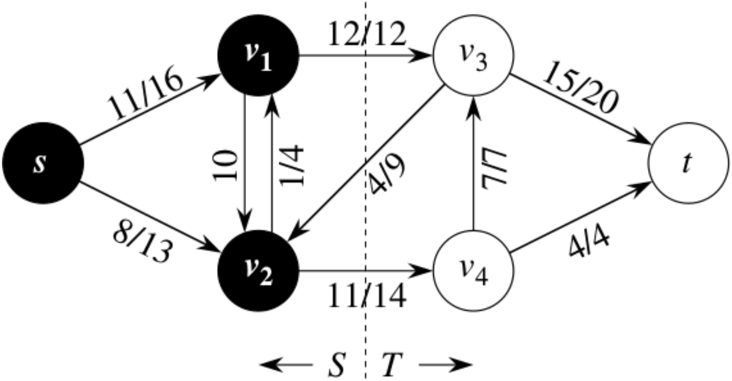
Cuts of flow networks

- A **cut** (S, T) of a flow network is a partition of its vertex set V into a set S (containing the source vertex s) and a set $T = V - S$ (containing the sink vertex t).



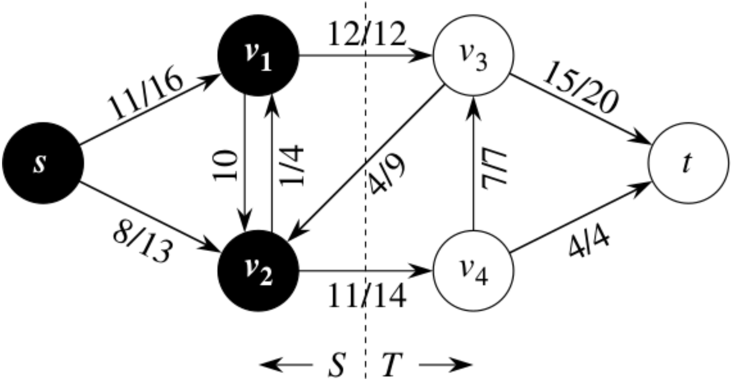
Net flow across the cut

- The net flow across the cut is the total sum of flows on edges going from S to T minus the total sum of edges coming from T to S. In the example below, the net flow is $12 + 11 - 4 = 19$.



Capacity of the cut

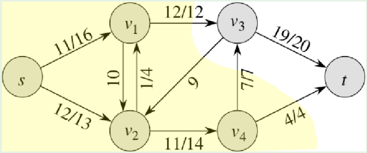
- The capacity of the cut is the sum total of the capacities of the edges leaving the cut from S to T (ignore the edges coming into S from T). In the example below, the capacity of the cut is $12 + 14 = 26$.



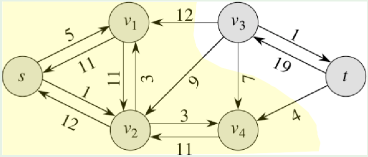
Minimum cut

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

G(flow value $= 23$)



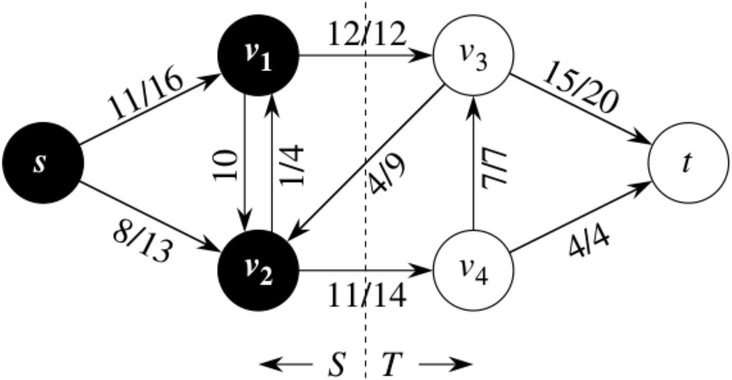
Corresponding $G_{residual}$



Two key observations

Observation 1

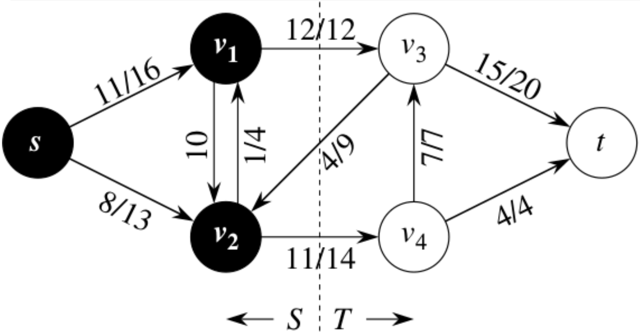
For any cut (S, T) of the flow network, the net flow of the cut is **equal to** the flow value of the network. (To be proved in the lecture)



Two key observations

Observation 2

Any flow value of a network is always $\leq$ the capacity of any cut. (To be proved in the lecture.)



Consequence of observation 2

The **minimum (capacity) cut** gives an **upper bound** on the **maximum flow** in the network G.

Min-cut Max-flow theorem

If f is some flow (assignment) of a network and (S, T) is some cut such that:

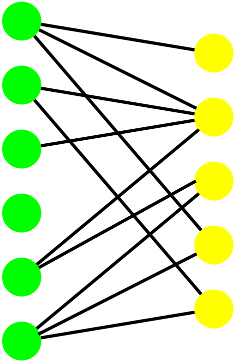
$$value(f) = capacity(S, T)$$

then (S, T) is the **minimum (capacity) cut**, and, equivalently, f is **maximum flow** in the flow network.

(To be proven in the lecture)

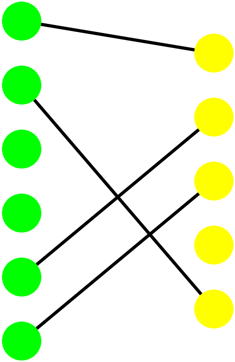
Application of Max Flow problem: Maximum Cardinality Bipartite **matching**

A Bipartite Graph



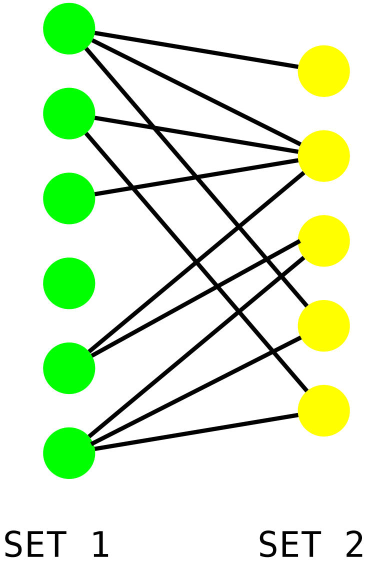
SET 1 SET 2

One possible matching in a Bipartite Graph

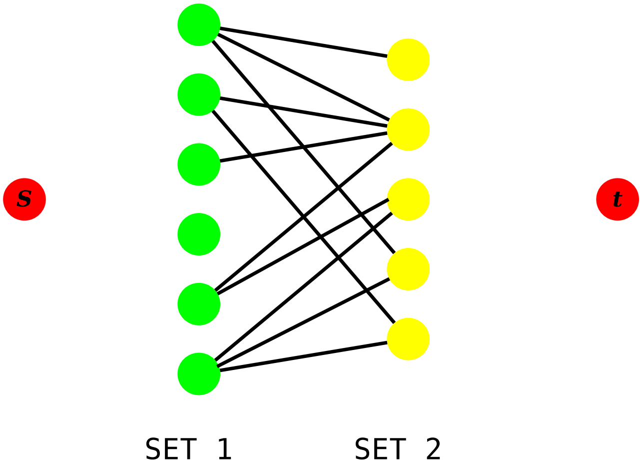


SET 1 SET 2

Converting maximum cardinality bipartite matching into max flow problem

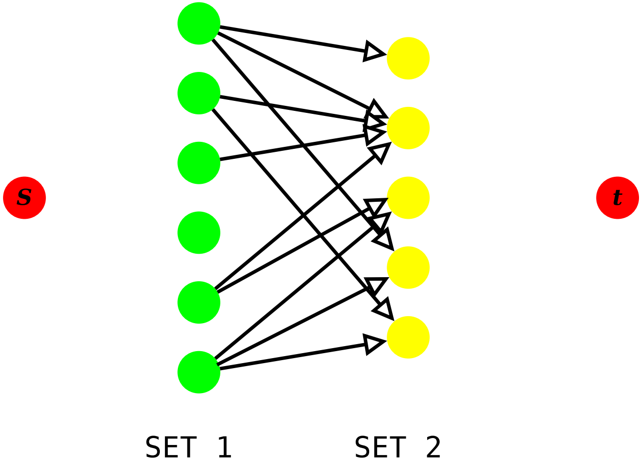


Converting maximum cardinality bipartite matching into max flow problem



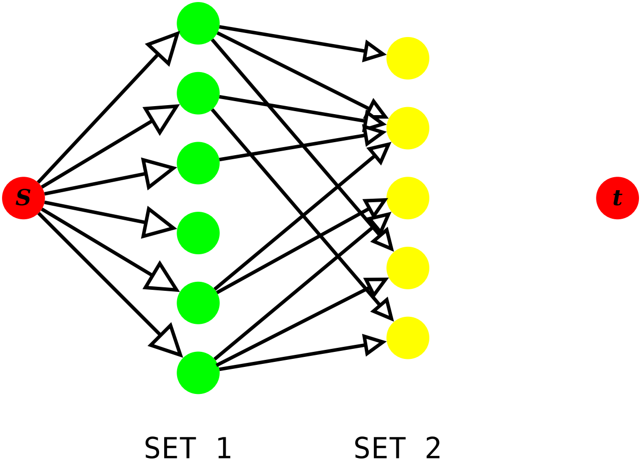
Add dummy source (s) and sink (t) vertices.

Converting maximum cardinality bipartite matching into max flow problem



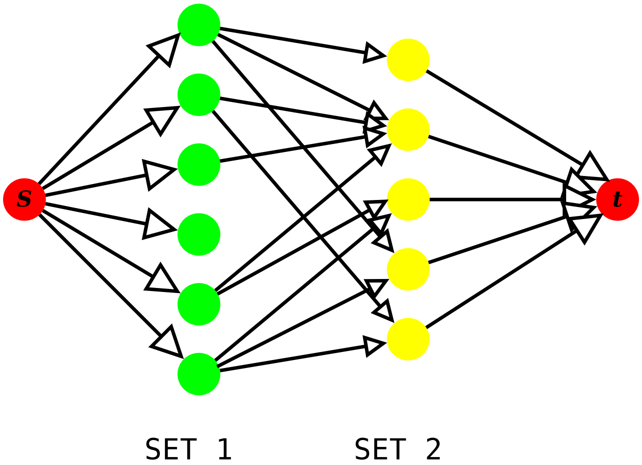
Convert undirected edges to **directed** edges going from set 1 to set 2.

Converting maximum cardinality bipartite matching into max flow problem



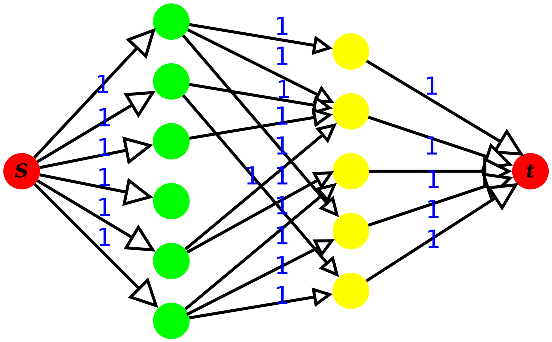
Add new **directed** edges going from source (s) to each vertex in set 1.

Converting maximum cardinality bipartite matching into max flow problem



Add new **directed** edges going from each vertex in set 2 to (t).

Converting maximum cardinality bipartite matching into max flow problem



Assign edge **capacities** of 1 to each edge.

On this graph run max-flow computation. The resultant edges between set 1 and 2 with a flow assignment of 1 defines a maximum cardinality bipartite matching

Summary

- Many real-life problems can be casted as max-flow/min-cut problems
- Max-flow and Min-cut are two-sides of the same coin – solving one, solves another.
- In practice, min-cut problems are solved using max-flow algorithms, because min-cut $=$ max-flow theorem.
- Ford-Fulkerson algorithm originally proposed the max-flow problem in pseudo-polynomial time.
- Dinic/Edmond-Karp proposed specific implementations of Ford-Fulkerson halt after polynomial number of iterations, independent of the actual edge capacities.

-=o0o=-

END

-=o0o=-