دانشگاه آزاد اسلامی واحد تبریز

نام درس: طراحی و تحلیل الگوریتم های پیشرفته

بخش: حداکثر سازی جریان

نام استاد: دکتر مسعود کارگر

Maximum flow

- Main goals of the lecture:
  - *to understand how flow networks and maximum flow problem can be* ***formalized****;*
  - *to understand the* ***Ford-Fulkerson*** *method and to be able to prove that it works correctly;*
  - *to understand the* ***Edmonds-Karp*** *algorithm and the intuition behind the analysis of its worst-case running time.*
  - *to be able to apply the Ford-Fulkerson method to solve the* ***maximum-bipartite-matching*** *problem.*

Flow networks

- *What if weights in a graph are maximum capacities of some flow of material?*
  - Pipe network to transport fluid (e.g., water, oil)
    - Edges – pipes, vertices – junctions of pipes
  - Data communication network
    - Edges – network connections of different capacity, 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

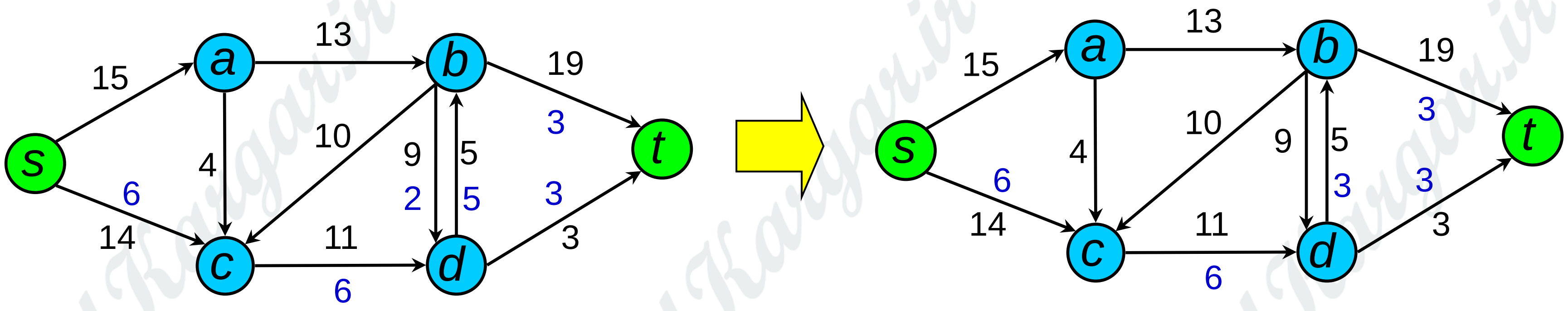
- *How do we formalize flows?*
  - Graph $G=(V,E)$ – a **flow network**
    - Directed, each edge has **capacity** $c(u,v) \geq 0$
    - Two special vertices: ***source*** *s*, and ***sink*** *t*
    - For any other vertex *v*, there is a path $s \to \ldots \to v \to \ldots \to t$
  - **Flow** – a function $f : V \times V \to \mathbf{R}$
    - *Capacity constraint*: For all $u, v \in V$: $f(u,v) \leq c(u,v)$
    - *Skew symmetry*: For all $u, v \in V$: $f(u,v) = -f(v,u)$
    - *Flow conservation:* For all $u \in V - \{s, t\}$:

$$\sum_{v \in V} f(u,v) = f(u,V) = 0, \text{ or}$$

$$\sum_{v \in V} f(v,u) = f(V,u) = 0$$

Cancellation of flows

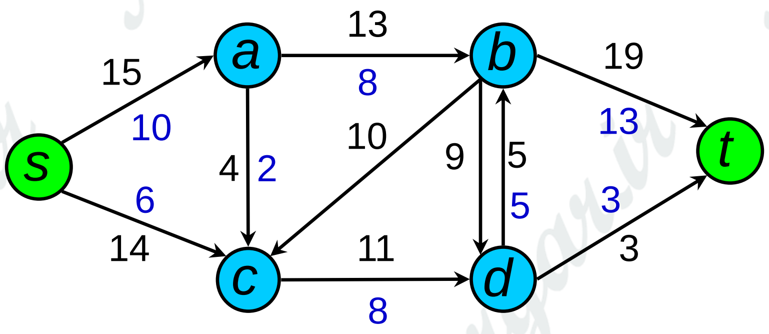
- *Do we want to have positive flows going in both directions between two vertices?*
  - No! such flows *cancel* (maybe partially) each other
  - Skew symmetry – notational convenience



Maximum flow

- *What do we want to maximize?*
  - ***Value*** of the flow *f*:

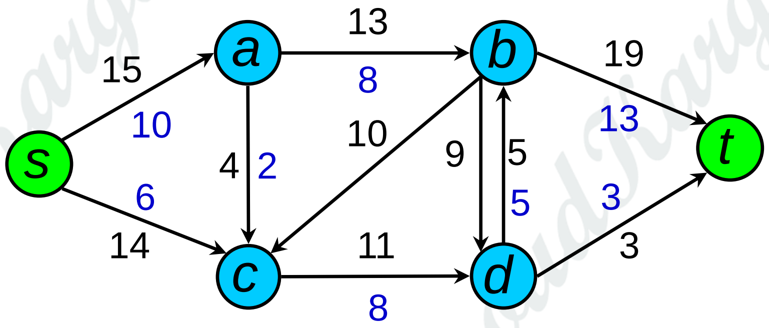
$$|f| = \sum_{v \in V} f(s,v) = f(s,V) = f(V,t)$$



- *We want to find a flow of maximum value!*

Augmenting path

- *Idea for the algorithm*:
  - If we have some flow,…
  - …and can find a path *p* from *s* to *t* (***augmenting path***), such that there is *a* > 0, and for each edge (*u,v*) in *p* we can add *a* units of flow: $f(u,v) + a \leq c(u,v)$
  - Then just do it, to get a better flow!
  - *Augmenting path in this graph*?



The Ford-Fulkerson method

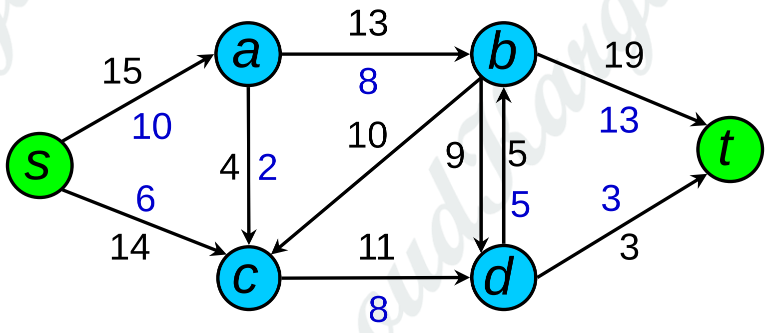
- Sketch of the method:

```
Ford-Fulkerson(G,s,t)
initialize flow f to 0 everywhere
while there is an augmenting path p do
    augment flow f along p
return f
```

- *How do we find augmenting path?*
- *How much additional flow can we send through that path?*
- *Does the algorithm always find the maximum flow?*

Residual network

- *How do we find augmenting path?*
  - It is any path in ***residual network***:
    - Residual capacities: $c_f(u,v) = c(u,v) - f(u,v)$
    - Residual network: $G_f=(V,E_f)$, where
      $E_f = \{(u,v) \in V \times V : c_f(u,v) > 0\}$
    - *What happens when $f(u,v) < 0$ (and $c(u,v) = 0$)?*
    - Observation – edges in E_f are either edges in E or their reversals: $|E_f| \leq 2|E|$
- *Compute residual network:*



Residual capacity of a path

- *How much additional flow can we send through an augmenting path?*
  - *Residual capacity* of a path p in G_f:
  - $c_f(p) = \min\{c_f(u,v): (u,v) \text{ is in } p\}$
  - Doing augmentation: for all (u,v) in p, we just add this $c_f(p)$ to $f(u,v)$ (and subtract it from $f(v,u)$)
  - Resulting flow is a valid flow with a larger value.
  - *What is the residual capacity of the path* (s,a,b,t)?

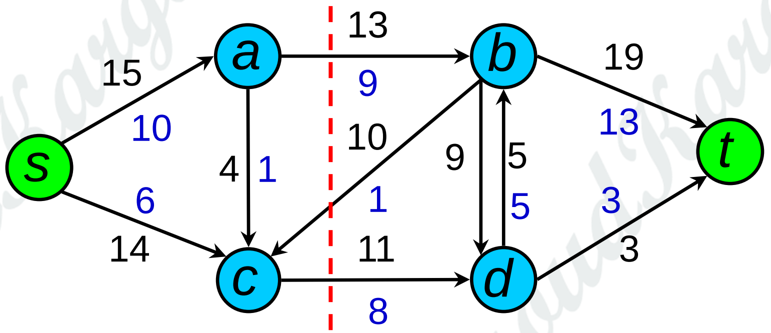
The Ford-Fulkerson method

```
Ford-Fulkerson(G,s,t)
01 for each edge (u,v) in G.E do
02     f(u,v) ← f(v,u) ← 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) is in p}
05     for each edge (u,v) in p do
06         f(u,v) ← f(u,v) + c_f
07         f(v,u) ← -f(u,v)
08 return f
```

- The algorithms based on this method differ in how they choose *p* in step 03.

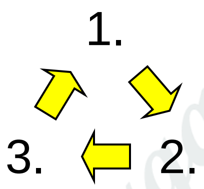
Cuts

- *Does it always find the maximum flow?*
  - 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*)) through the cut is the sum of flows *f*(*u*,*v*), where *u* ∈ *S and v* ∈ *T*
  - The ***capacity*** (*c*(*S*,*T*)) of the cut sum of capacities *c*(*u*,*v*), where *u* ∈ *S and v* ∈ *T*
  - ***Minimum cut*** – a cut with the smallest capacity of all cuts
  - |*f*|= *f*(*S*,*T*)



Correctness of Ford-Fulkerson

- *Max-flow min-cut theorem*:
  - If *f* is the flow in G, the following conditions a re 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
  - We have to prove three parts:



  - From this we have 1.⇔2., which means that the Ford-Fulkerson method always correctly finds a maximum flow

Worst-case running time

- *What is the worst-case running time of this method?*
  - Let's assume integer flows.
  - Each augmentation increases the value of the flow by some positive amount.
  - 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.
  - *Can we run into this worst-case*?
  - *Lesson: how an augmenting path is chosen is very important!*

Edmonds-Karp algorithm

- *Take **shortest path** (in terms of number of edges) as an augmenting path – Edmonds-Karp algorithm*
  - *How do we find such a shortest path?*
  - Running time $O(VE^2)$, because the number of augmentations is $O(VE)$
  - To prove this we need to prove that:
    - *The length of the shortest path does not decrease*
    - *Each edge can become **critical** at most ~ V/2 times.* Edge (u,v) on an augmenting path p is critical if it has the minimum residual capacity in the path:

      $c_f(u,v) = c_f(p)$

Non-decreasing shortest paths

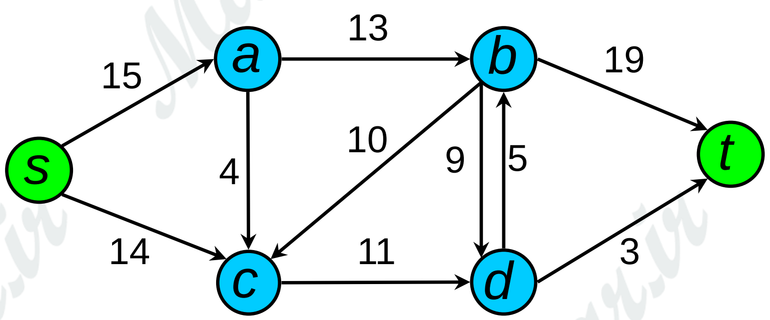
- *Why does the length of a shortest path from s to any v does not decrease?*
  - Observation: Augmentation may add some edges to residual network or remove some.
  - Only the added edges ("shortcuts") may potentially decrease the length of a shortest path.
  - Let's supose (*s*,…,*v*) – the shortest decreased-length path and let's derive a contradiction

Number of augmentations

- *Why each edge can become critical at most ~V/2 times?*
  - Scenario for edge (*u*,*v*):
    - Critical the first time: (*u*,*v*) on an augmenting path
    - Disappears from the network
    - Reappears on the network: (*v*,*u*) has to be on an augmenting path
    - We can show that in-between these events the distance from *s* to *u* increased by at least 2.
    - This can happen at most *V*/2 times
- We have proved that the running time of Edmonds-Karp is $O(VE^2)$.

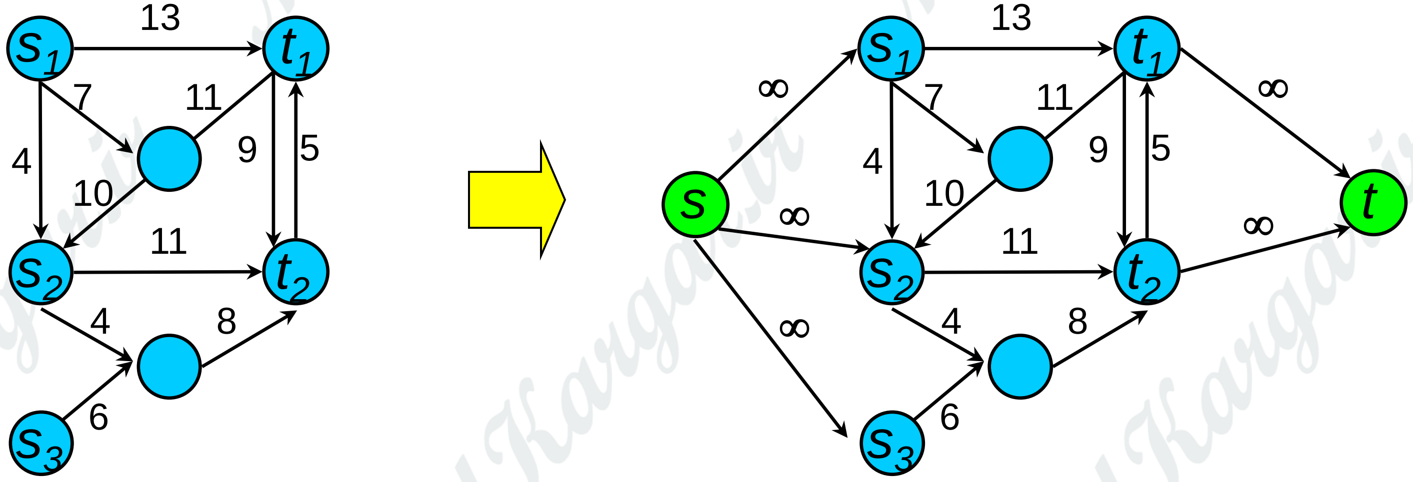
Example of Edmonds-Karp

- *Run the Edmonds-Karp algorithm on the following graph:*



Multiple sources or sinks

- *What if we have more sources or sinks?*
  - Augment the graph to make it with one source and one sink!



Application of max-flow

- *Maximum bipartite matching problem*
  - ***Matching*** in a graph is a subset *M* of edges such that each vertex has at most one edge of *M* incident on it. It puts vertices in pairs.
  - We look for *maximum* matching in a ***bipartite*** graph, where $V = L \cup R$, *L* and *R* are disjoint and all edges go between *L* and *R*
  - Dating agency example:
    - *L* – women, *R* – men.
    - An edge between vertices: they have a chance to be “compatible” (can be matched)
    - Do as many matches between “compatible” persons as possible

Maximum bipartite matching

- *How can we reformulate this problem to become a max-flow problem?*

- *What is the running time of the algorithm if we use the Ford-Fulkerson method?*