# Discrete Mathematics

CS204: Spring, 2008

**Jong C. Park**
**Computer Science Department**
**KAIST**

Today's Topics

Introduction

A Maximal Flow Algorithm

The Max Flow, Min Cut Theorem

Matching

# NETWORK MODELS

# Introduction

- Definition
  - A transport network (or more simply network) is a simple, weighted, directed graph satisfying:
    - (a) A designated vertex, the source, has no incoming edges.
    - (b) A designated vertex, the sink, has no outgoing edges.
    - (c) The weight $C_{ij}$ of the directed edge $(i, j)$, called the capacity of $(i, j)$, is a nonnegative number.
- Example
  - the graph of Figure 10.1

# Introduction

- Definition
  - Let $G$ be a transport network. Let $C_{ij}$ denote the capacity of the directed edge $(i, j)$. A flow $F$ in $G$ assigns each directed edge $(i, j)$ a nonnegative number $F_{ij}$ such that:
    (a) $F_{ij} \le C_{ij}$.
    (b) For each vertex $j$, which is neither the source nor the sink,
    $\Sigma_i F_{ij} = \Sigma_i F_{ji}$ (* property of the conservation of flow)
    - In a sum such as (*), unless specified otherwise, the sum is assumed to be taken over all vertices $i$. Also, if $(i, j)$ is not an edge, we set $F_{ij} = 0$.
  - We call $F_{ij}$ the flow in edge $(i, j)$. For any vertex $j$, we call $\Sigma_i F_{ij}$ the flow into $j$ and we call $\Sigma_i F_{ji}$ the flow out of $j$.

# Introduction

- Example
  - Sample flow
    - $F_{ab} = 2$, $F_{bc} = 2$, $F_{cz} = 3$, $F_{ad} = 3$, $F_{dc} = 1$, $F_{de} = 2$, $F_{ez} = 2$

# Introduction

- Theorem
  - Given a flow $F$ in a network, the flow out of the source $a$ equals the flow into the sink $z$, that is,

    $$\Sigma_i F_{ai} = \Sigma_i F_{iz}.$$

  - Proof.
    - Let $V$ be the set of vertices.
    - We have

      $$\Sigma_{j\in V}(\Sigma_{i\in V} F_{ij}) = \Sigma_{j\in V}(\Sigma_{i\in V} F_{ji}),$$

      since each double sum is $\Sigma_{e\in E} F_e$, where $E$ is the set of edges.

# Introduction

- Now, $0 = \Sigma_{j\in V}(\Sigma_{i\in V} F_{ij} - \Sigma_{i\in V} F_{ji})$

  $= (\Sigma_{i\in V} F_{iz} - \Sigma_{i\in V} F_{zi}) +$

  $(\Sigma_{i\in V} F_{ia} - \Sigma_{i\in V} F_{ai}) +$

  $\Sigma_{j\in V, j\neq a,z}(\Sigma_{i\in V} F_{ij} - \Sigma_{i\in V} F_{ji})$

  $= \Sigma_{i\in V} F_{iz} - \Sigma_{i\in V} F_{ai}$

  since $F_{zi} = 0 = F_{ia}$, for all $i \in V$, and (by definition)

  $\Sigma_{i\in V} F_{ij} - \Sigma_{i\in V} F_{ji} = 0$ if $j \in V - \{a,z\}$.

# Introduction

- Definition
  - Let $F$ be a flow in a network $G$. The value

    $$\Sigma_i F_{ai} = \Sigma_i F_{iz}$$

    is called the value of the flow $F$.
- Examples
  - the value of the flow in the network of Figure 10.1.2
  - A Pumping Network
    - Figure 10.1.3
    - supersource, supersink
  - A Traffic Flow Network

# A Maximal Flow Algorithm

- Note
  - If $G$ is a transport network, a maximal flow in $G$ is a flow with maximal value.
  - Consider the edges of $G$ to be undirected and let

    $$P = (v_0, v_1, \ldots, v_n),\ v_0 = a,\ v_n = z$$

    be a path from $a$ to $z$ in this undirected graph.
    - If an edge $e$ in $P$ is directed from $v_{i-1}$ to $v_i$, we say that

      $e$ is properly oriented (with respect to $P$);

      otherwise, we say that

      $e$ is improperly oriented (with respect to $P$).

# A Maximal Flow Algorithm

- Example
  - the path from *a* to *z* in Figure 10.2.2
  - after increasing the flow by 1 (Figure 10.2.3)
  - the four possible orientations of the edges incident on *x*
    - Figure 10.2.4
- Example
  - the path from *a* to *z* in Figure 10.2.5
  - after increasing the flow by 1 (Figure 10.2.6)

# A Maximal Flow Algorithm

- Theorem
  - Let $P$ be a path from $a$ to $z$ in a network $G$ satisfying the following conditions:
    - (a) For each properly oriented edge $(i, j)$ in $P$, $F_{ij} < C_{ij}$.
    - (b) For each improperly oriented edge $(i, j)$ in $P$, $0 < F_{ij}$.
  - Let $\Delta = \min X$, where $X$ consists of the number $C_{ij} - F_{ij}$, for properly oriented edges $(i, j)$ in $P$, and $F_{ij}$, for improperly oriented edges $(i, j)$ in $P$.

# A Maximal Flow Algorithm

– Define

$F^*_{ij} =$ $F_{ij}$ if $(i, j)$ is not in $P$,

$F_{ij} + \Delta$ if $(i, j)$ is properly oriented in $P$, and

$F_{ij} - \Delta$ if $(i, j)$ is not properly oriented in $P$.

– Then $F^*$ is a flow whose value is $\Delta$ greater than the value of $F$.

# A Maximal Flow Algorithm

- Procedure
  - Start with a flow (e.g., the flow in which the flow in each edge is 0).
  - Search for a path satisfying the conditions of the earlier theorem.
    - If no such path exists, stop; the flow is maximal.
  - Increase the flow through the path by $\Delta$, where $\Delta$ is defined as in the earlier theorem, and go to line 2.

# A Maximal Flow Algorithm

**Algorithm 10.2.4: Finding a Maximal Flow in a Network**

Input: A network with source $a$, sink $z$, capacity $C$, vertices $a = v_0, \ldots, v_n = z$, and $n$

Output: A maximal flow $F$

```
max_flow(a, z, C, v, n) {
  // v's label is (predecessor(v), val(v))
  // start with zero flow
  for each edge (i, j)
    F_ij = 0
  while (true) {
    // remove all labels
    for i = 0 to n {
      predecessor(v_i) = null
      val(v_i) = null
    }
    // label a
    predecessor(a) = —
    val(a) = ∞
    // U is the set of unexamined, labeled vertices
    U = {a}
```

# A Maximal Flow Algorithm

**Algorithm 10.2.4 (continued)**

```
// continue until z is labeled
while (val(z) == null) {
   if (U == ∅) // flow is maximal
      return F
   choose v in U
   U = U − {v}
   Δ = val(v)
   for each edge (v, w) with val(w) == null
      if (F_vw < C_vw) {
         predecessor(w) = v
         val(w) = min{Δ, C_vw − F_vw}
         U = U ∪ {w}
      }
   for each edge (w, v) with val(w) == null
      if (F_wv > 0) {
         predecessor(w) = v
         val(w) = min{Δ, F_wv}
         U = U ∪ {w}
      }
} // end while (val(z) == null) loop
```

# A Maximal Flow Algorithm

**Algorithm 10.2.4 (continued)**

```
        // find path P from a to z on which to revise flow
        w_0 = z
        k = 0
        while (w_k ¬= a) {
            w_{k+1} = predecessor(w_k)
            k = k + 1
        }
        P = (w_{k+1}, w_k, ..., w_1, w_0)
        Δ = val(z)
        for i = 1 to k + 1 {
            e = (w_i, w_{i-1})
            if (e is properly oriented in P)
                F_e = F_e + Δ
            else
                F_e = F_e − Δ
        }
    } // end while (true) loop
}
```

# Summary

- Introduction
- A Maximal Flow Algorithm
- The Max Flow, Min Cut Theorem
- Matching