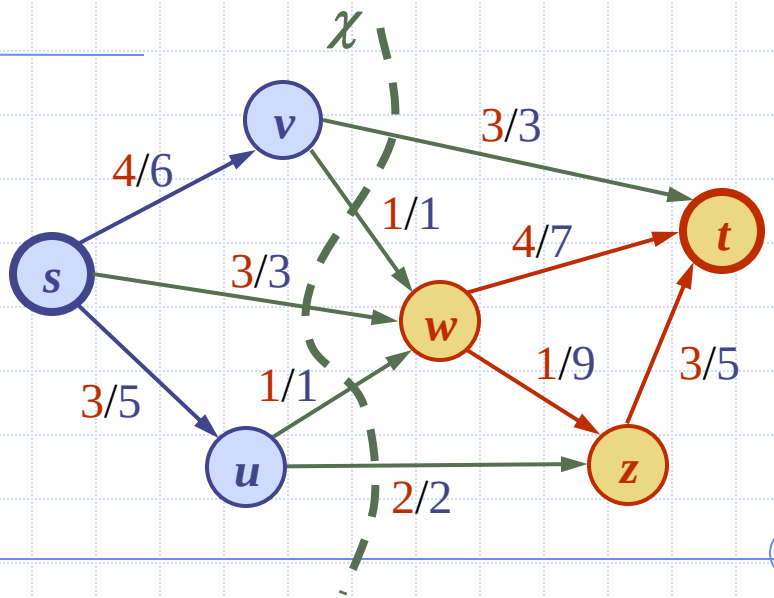
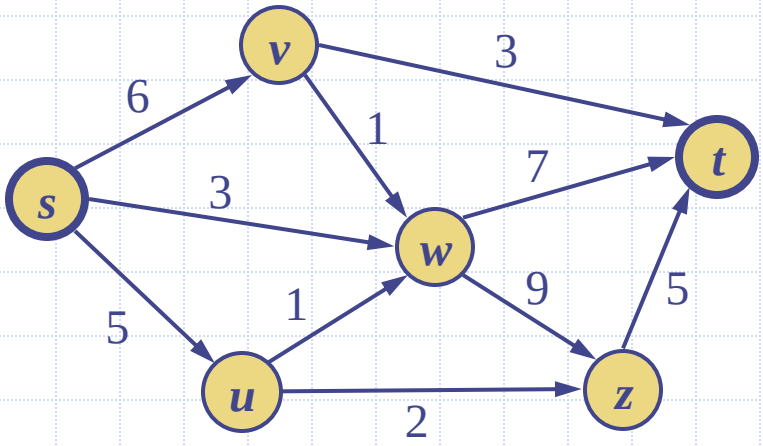
Presentation for use with the textbook, Algorithm Design and Applications, by M. T. Goodrich and R. Tamassia, Wiley, 2015

# Μέγιστη Ροή (Maximum Flow)



# Ροή Δικτύου (Network Flow)

- Ένα δίκτυο ροής (ή απλά δίκτυο) $N$ αποτελείται από:
  - Ένα σταθμισμένο κατευθυνόμενο γράφο $G$ με μη αρνητικές ακέραιες τιμές ως βάρη ακμών, όπου το βάρος κάθε ακμής $e$ καλείται χωρητικότητα $c(e)$ του $e$.
  - Δύο συγκεκριμένες κορυφές $s$ και $t$ του $G$, ονομάζονται πηγή (source) και καταβόθρα (sink), αντίστοιχα, έτσι ώστε η $s$ να μην έχει εισερχόμενες ακμές και η $t$ να μην έχει εξερχόμενες ακμές.
- Παράδειγμα:



# Ροή (Flow)

- Μια ροή $f$ για ένα δίκτυο $N$ είναι η ανάθεση ακέραιας τιμής $f(e)$ σε κάθε ακμή $e$ που ικανοποιεί τους ακόλουθους κανόνες:

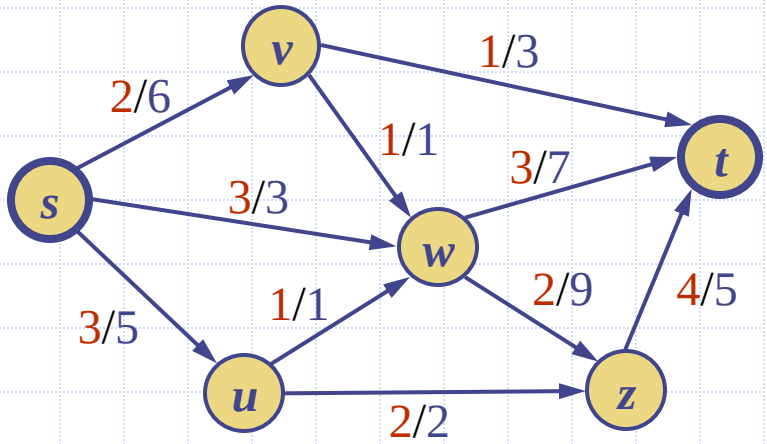
  Κανόνας χωρητικότητας (capacity rule): Για κάθε ακμή $e$, $0 \leq f(e) \leq c(e)$

  Κανόνας διατήρησης (conservation rule): Για κάθε κορυφή $v \neq s, t$

$$\sum_{e \in E^-(v)} f(e) = \sum_{e \in E^+(v)} f(e)$$

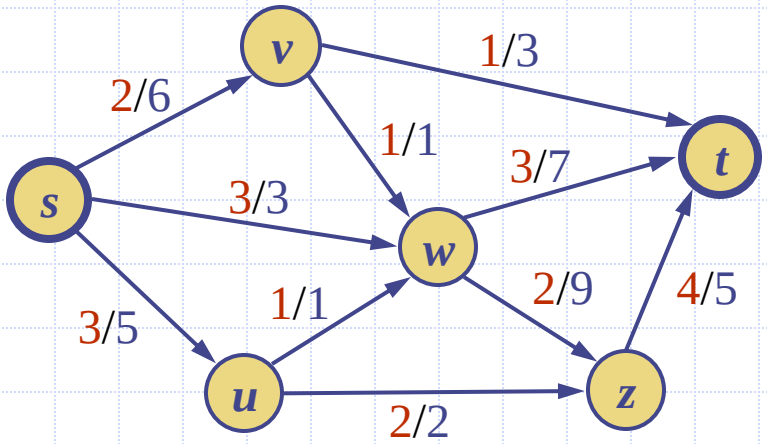
  όπου $E^-(v)$ και $E^+(v)$ είναι οι εισερχόμενες και οι εξερχόμενες ακμές της $v$, αντίστοιχα.

- Η τιμή ροής $f$, που συμβολίζεται με $|f|$, είναι η συνολική ροή από το source, και είναι η ίδια με τη συνολική ροή στο sink.
- Παράδειγμα:

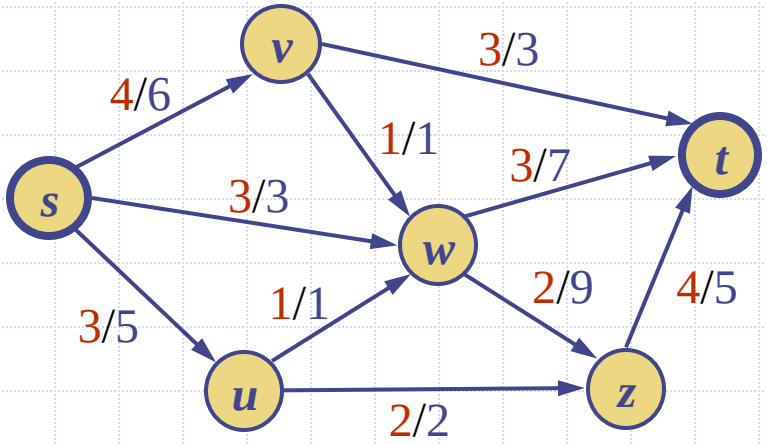


# Μέγιστη Ροή (Maximum Flow)

- Μια ροή σε ένα δίκτυο $N$ λέγεται ότι είναι μέγιστη αν η τιμή της είναι η μεγαλύτερη από όλες τις πιθανές ροές του $N$.
- Το πρόβλημα μέγιστης ροής συνίσταται στην εύρεση μιας μέγιστης ροής για ένα δεδομένο δίκτυο $N$.
- Εφαρμογές
  - Υδραυλικά συστήματα
  - Ηλεκτρικά κυκλώματα
  - Κυκλοφοριακή κίνηση
  - Μεταφορά εμπορευμάτων



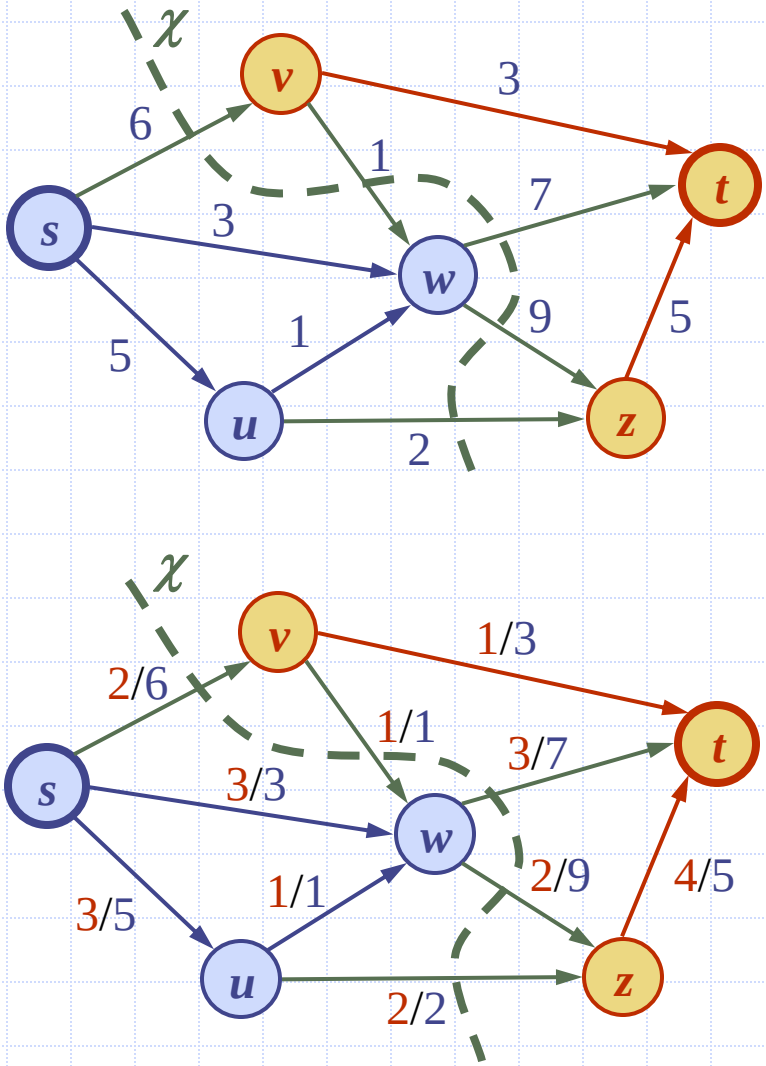

Ροή με τιμή 8 = 2 + 3 + 3 = 1 + 3 + 4




Μέγιστη ροή με τιμή 10 = 4 + 3 + 3 = 3 + 3 + 4

# Τομή (Cut)

- Μια τομή ενός δικτύου $N$ με source $s$ και sink $t$ είναι μια διαμέριση (partition) $\chi = (V_s, V_t)$ των κορυφών του $N$ έτσι ώστε $s \in V_s$ και $t \in V_t$
  - Μια προς τα εμπρός ακμή (forward edge) της τομής $\chi$: αφετηρία στο $V_s$ και προορισμό στο $V_t$
  - Μια προς τα πίσω ακμή (backward edge) της τομής $\chi$: αφετηρία στο $V_t$ και προορισμό στο $V_s$
- Ροή $f(\chi)$ διαμέσου της τομής $\chi$: συνολική ροή των προς τα εμπρός ακμών μείον τη συνολική ροή των προς τα πίσω ακμών.
- Χωρητικότητα $c(\chi)$ της τομής $\chi$: η συνολική χωρητικότητα των προς τα εμπρός ακμών.
- Παράδειγμα:
  - $c(\chi) = 24 = 6 + 7 + 9 + 2$
  - $f(\chi) = 8 = 2 - 1 + 3 + 2 + 2$



# Ροές και Τομές

Λήμμα:

Η ροή $f(\chi)$ διαμέσου οποιασδήποτε τομής $\chi$ είναι ίση με την τιμή της ροής $|f|$.

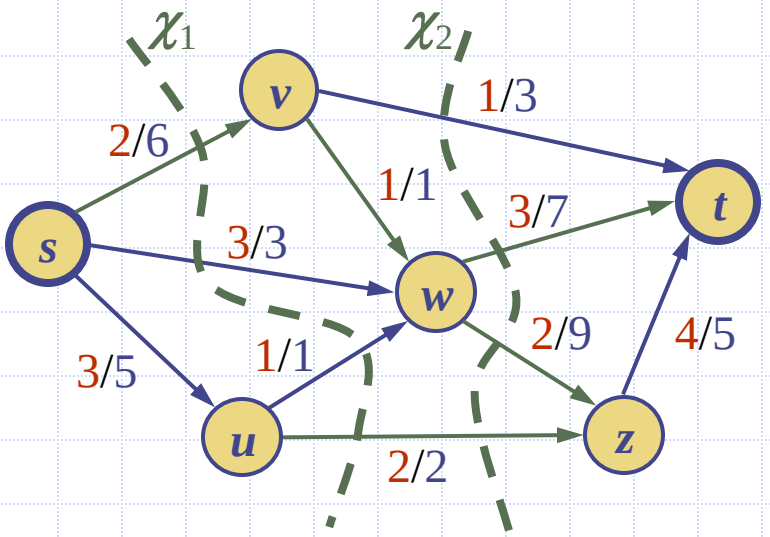
Λήμμα:

Η ροή $f(\chi)$ διαμέσου της τομής $\chi$ είναι μικρότερη ή ίση από την χωρητικότητα $c(\chi)$ της τομής.

Θεώρημα:

Η τιμή οποιασδήποτε ροής είναι μικρότερη ή ίση από τη χωρητικότητα οποιασδήποτε τομής, δλδ., για κάθε ροή $f$ και κάθε τομή $\chi$, ισχύει

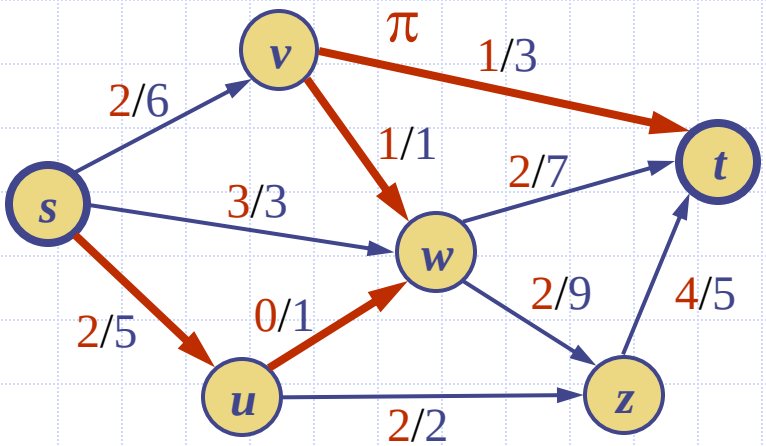
$$|f| \leq c(\chi)$$




$$c(\chi_1) = 12 = 6 + 3 + 1 + 2$$
$$c(\chi_2) = 21 = 3 + 7 + 9 + 2$$
$$|f| = 8$$

# Επαυξημένη διαδρομή (augmenting path)

- Για μια ροή $f$ σε ένα δίκτυο $N$
- Έστω $e$ μια ακμή από το $u$ στο $v$:
  - Υπολειπόμενη χωρητικότητα της $e$ από το $u$ στο $v$: $\Delta_f(u, v) = c(e) - f(e)$
  - Υπολειπόμενη χωρητικότητα της $e$ από το $v$ στο $u$: $\Delta_f(v, u) = f(e)$
- Έστω $\pi$ μια διαδρομή από το $s$ στο $t$
  - Η υπολειπόμενη χωρητικότητα $\Delta_f(\pi)$ της διαδρομής $\pi$ είναι η μικρότερη από τις υπολειπόμενες χωρητικότητες των ακμών της $\pi$ στην κατεύθυνση από το $s$ στο $t$
- Μια διαδρομή $\pi$ από το $s$ στο $t$ είναι μια επαυξημένη διαδρομή αν $\Delta_f(\pi) > 0$




$$\Delta_f(s,u) = 3$$
$$\Delta_f(u,w) = 1$$
$$\Delta_f(w,v) = 1$$
$$\Delta_f(v,t) = 2$$
$$\Delta_f(\pi) = 1$$
$$|f| = 7$$

# Επαύξηση Ροής (Flow Augmentation)

Λήμμα:

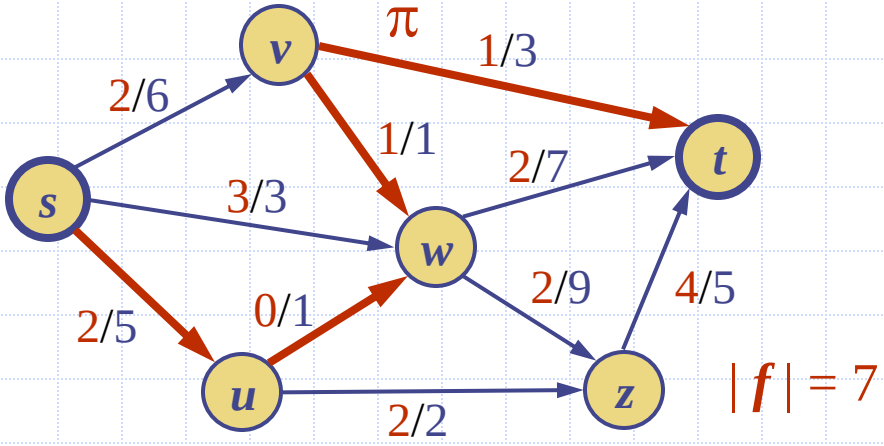
Έστω $\pi$ μια επαυξημένη διαδρομή για την ροή $f$ στο δίκτυο $N$. Υπάρχει μια ροή $f'$ για το $N$ με τιμή

$$|f'| = |f| + \Delta_f(\pi)$$

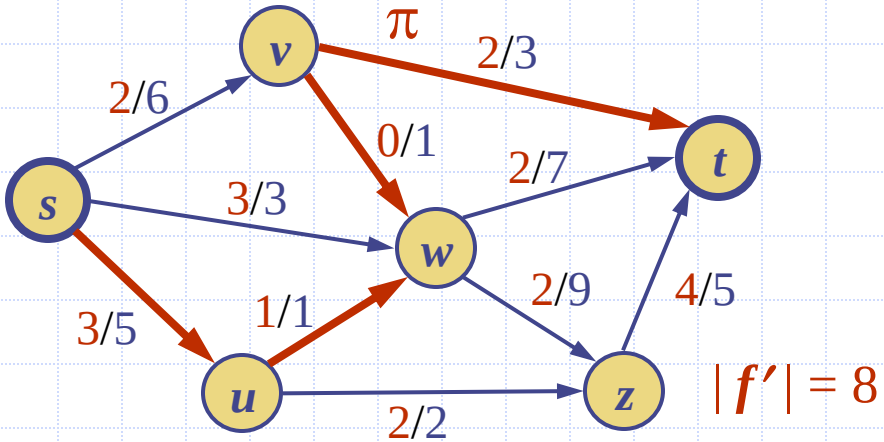
Απόδειξη:

Υπολογίζουμε την ροή $f'$ τροποποιώντας τη ροή των ακμών της $\pi$

- Ακμή προς τα εμπρός:
$$f'(e) = f(e) + \Delta_f(\pi)$$
- Ακμή προς τα πίσω:
$$f'(e) = f(e) - \Delta_f(\pi)$$




$\Delta_f(\pi) = 1$



# Ο Αλγόριθμος των Ford-Fulkerson

- Αρχικά, $f(e) = 0$ για κάθε ακμή $e$
- Επαναληπτικά:
  - Αναζήτηση μιας επαυξημένης διαδρομής $\pi$
  - Επαύξηση κατά $\Delta_f(\pi)$ της ροής μέσω των ακμών της $\pi$
- Μια εξειδικευμένη DFS (ή BFS) αναζητά μια επαυξημένη διαδρομή
  - Μια ακμή $e$ διατρέχεται από το $u$ στο $v$ εφόσον $\Delta_f(u, v) > 0$

**Algorithm** MaxFlowFordFulkerson($N$):
*Input:* Flow network $N = (G, c, s, t)$
*Output:* A maximum flow $f$ for $N$

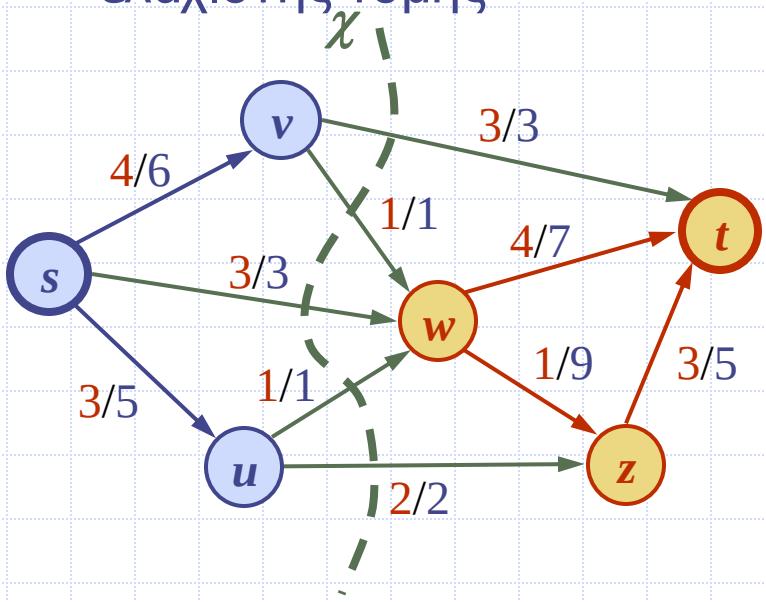
**for** each edge $e \in N$ **do**
  $f(e) \leftarrow 0$
*stop* $\leftarrow$ **false**
**repeat**
  traverse $G$ starting at $s$ to find an augmenting path for $f$
  **if** an augmenting path $\pi$ exists **then**
    // Compute the residual capacity $\Delta_f(\pi)$ of $\pi$
    $\Delta \leftarrow +\infty$
    **for** each edge $e \in \pi$ **do**
      **if** $\Delta_f(e) < \Delta$ **then**
        $\Delta \leftarrow \Delta_f(e)$
    **for** each edge $e \in \pi$ **do** // push $\Delta = \Delta_f(\pi)$ units along $\pi$
      **if** $e$ is a forward edge **then**
        $f(e) \leftarrow f(e) + \Delta$
      **else**
        $f(e) \leftarrow f(e) - \Delta$ // $e$ is a backward edge
  **else**
    *stop* $\leftarrow$ **true** // $f$ is a maximum flow
**until** *stop*

# Μέγιστη-Ροή (Max-Flow) και Ελάχιστη-Τομή (Min-Cut)

- Τερματισμός του αλγορίθμου των Ford-Fulkerson:
  - Όταν δεν υπάρχει επαυξημένη διαδρομή από το $s$ στο $t$ σε σχέση με την τρέχουσα ροή $f$
- Ορισμοί:
  - $V_s$ σύνολο των κορυφών που μπορούν να προσεγγιστούν από το $s$ με επαυξημένες διαδρομές
  - $V_t$ σύνολο κορυφών που απομένουν
- Η τομή $\chi = (V_s, V_t)$ έχει χωρητικότητα
  $$c(\chi) = |f|$$
  - Ακμή προς τα εμπρός: $f(e) = c(e)$
  - Ακμή προς τα πίσω: $f(e) = 0$
- Έτσι, η ροή $f$ έχει μέγιστη τιμή και η τομή $\chi$ έχει ελάχιστη χωρητικότητα

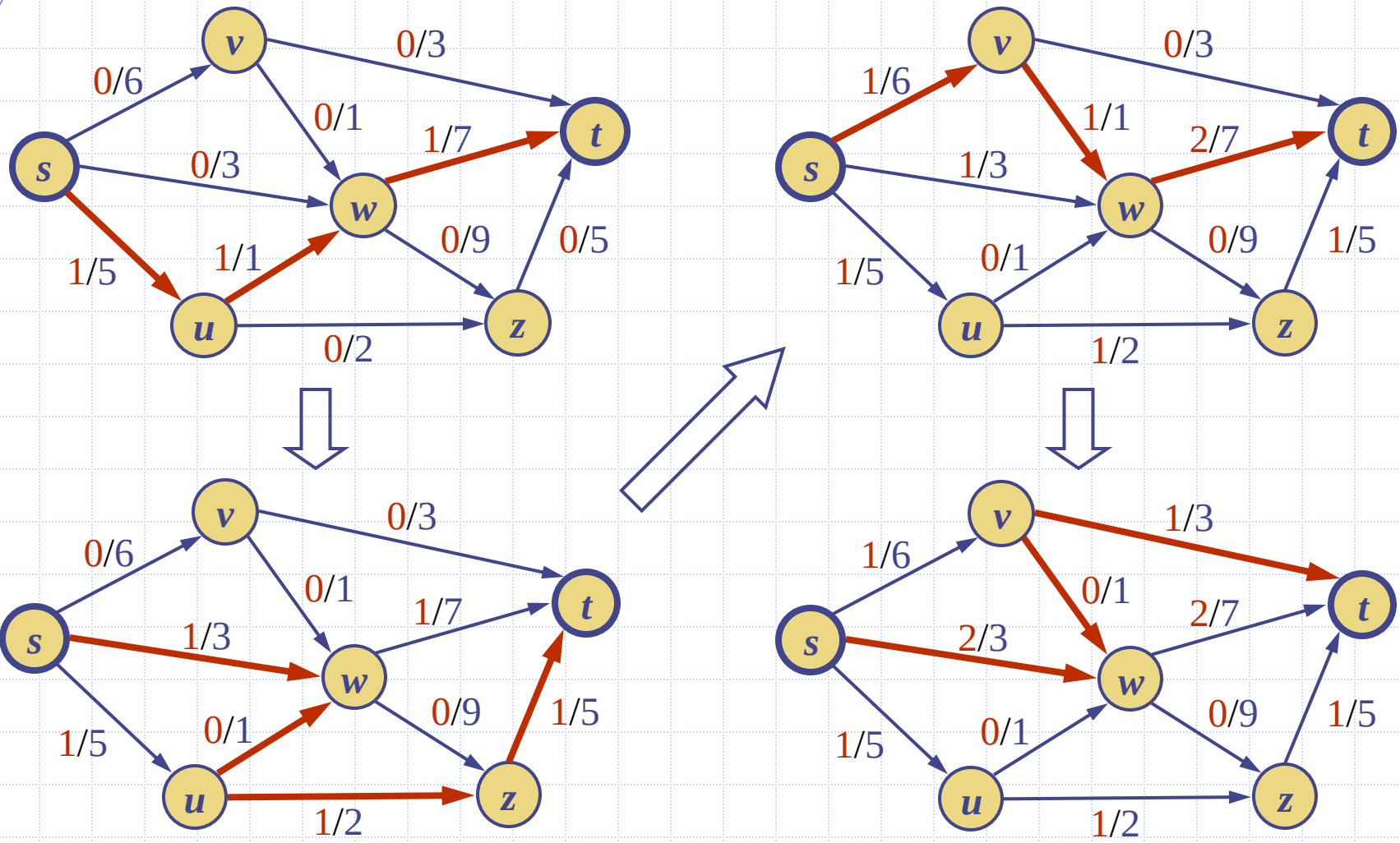
Θεώρημα:

Η τιμή της μέγιστης ροής ισούται με την χωρητικότητα μιας ελάχιστης τομής

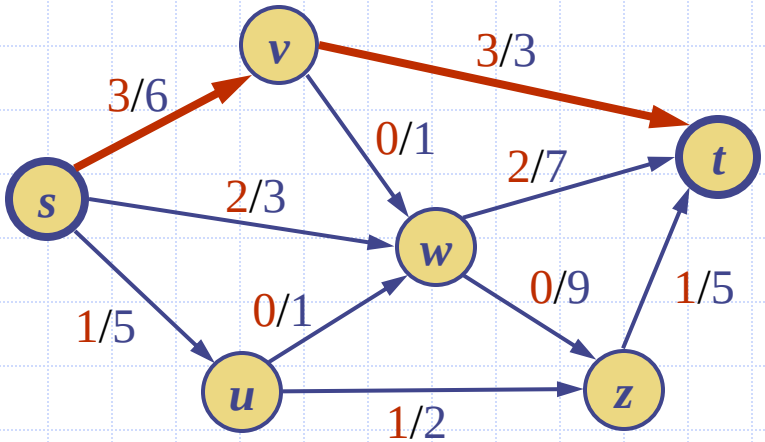


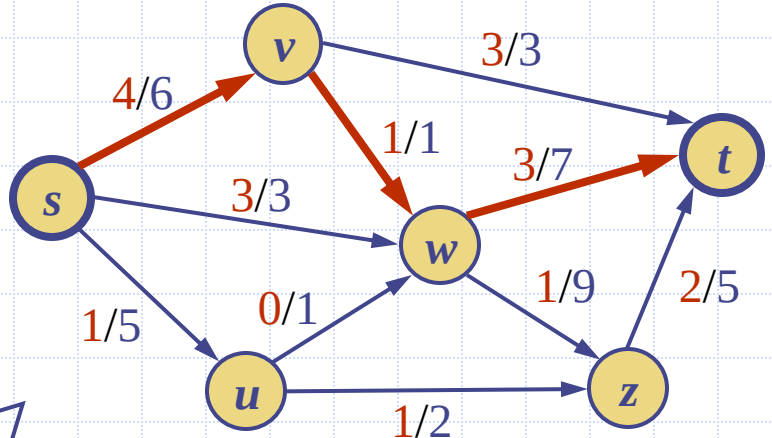

$$c(\chi) = |f| = 10$$

# Παράδειγμα (1)

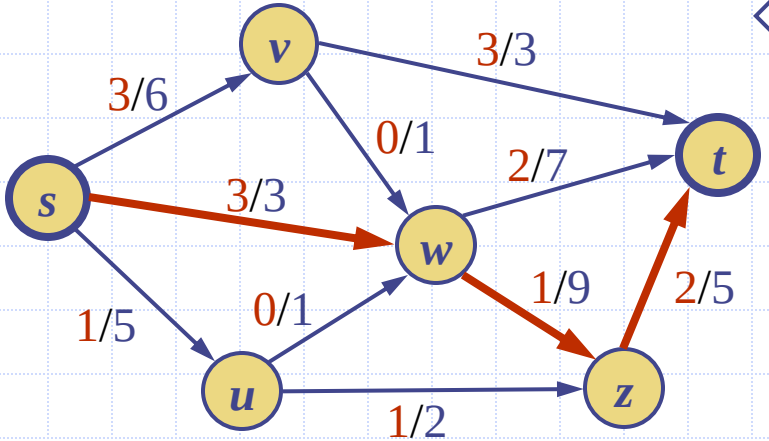


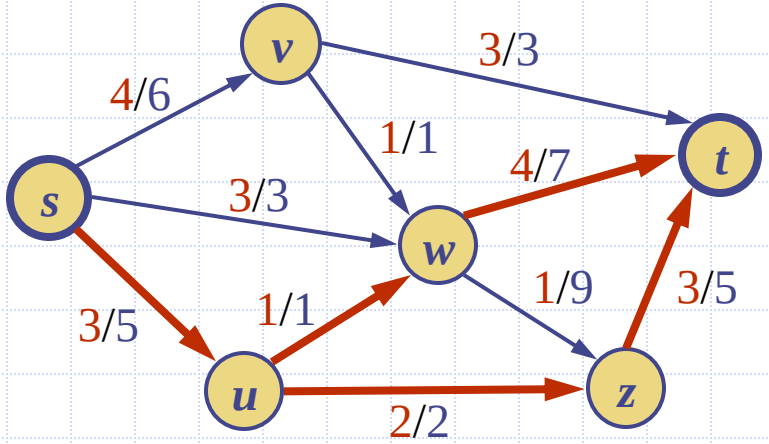
# Παράδειγμα (2)



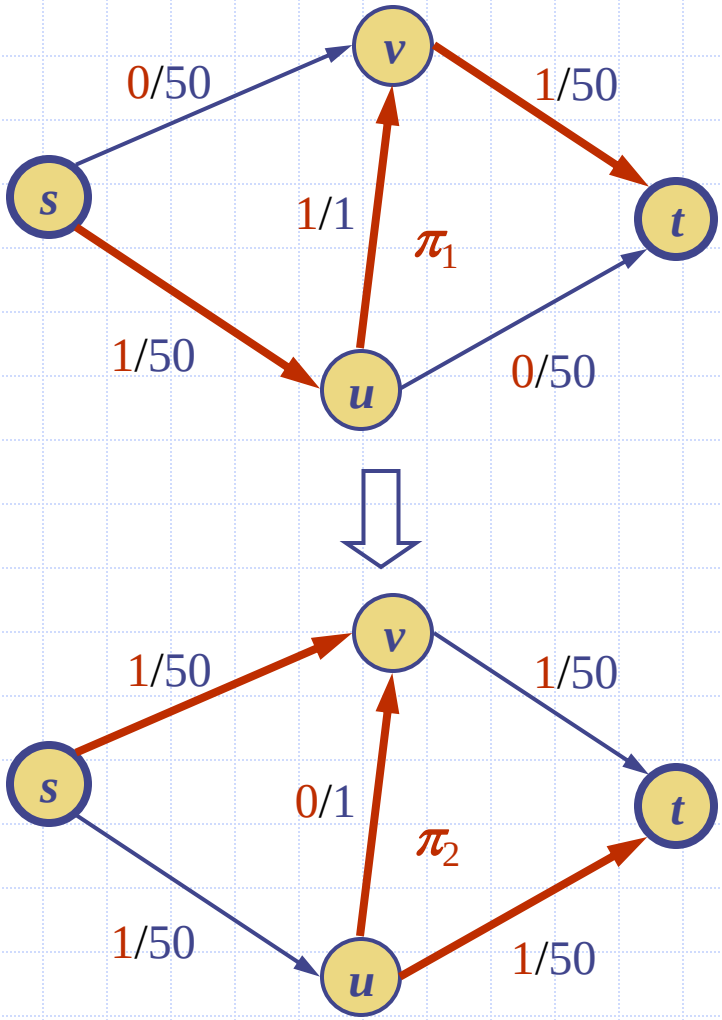

Δύο βήματα



# Ανάλυση

- Στην χειρότερη περίπτωση, ο αλγόριθμος των Ford-Fulkerson πραγματοποιεί $|f^*|$ επαυξήσεις ροών, όπου $f^*$ είναι μια μέγιστη ροή.
- Παράδειγμα:
  - Οι επαυξημένες διαδρομές βρίσκουν εναλλακτικές ανάμεσα στις $\pi_1$ και $\pi_2$
  - Ο αλγόριθμος πραγματοποιεί 100 επαυξήσεις
- Η εύρεση μιας επαυξημένης διαδρομής και η επαύξηση της ροής παίρνει $O(n+m)$ χρόνο
- Ο χρόνος εκτέλεσης του αλγορίθμου των Ford-Fulkerson' s είναι $O(|f^*|(n+m))$



# Maximum Bipartite Matching

- In the maximum bipartite matching problem, we are given a connected undirected graph with the following properties:
  - The vertices of G are partitioned into two sets, X and Y.
  - Every edge of G has one endpoint in X and the other endpoint in Y.
- Such a graph is called a **bipartite graph**.
- A **matching** in G is a set of edges that have no endpoints in common—such a set "pairs" up vertices in X with vertices in Y so that each vertex has at most one "partner" in the other set.
- The maximum bipartite matching problem is to find a matching with the greatest number of edges.

# Reduction to Max Flow

Let $G$ be a bipartite graph whose vertices are partitioned into sets $X$ and $Y$. We create a flow network $H$ such that a maximum flow in $H$ can be immediately converted into a maximum matching in $G$:

- We begin by including all the vertices of $G$ in $H$, plus a new source vertex $s$ and a new sink vertex $t$.
- Next, we add every edge of $G$ to $H$, but direct each such edge so that it is oriented from the endpoint in $X$ to the endpoint in $Y$. In addition, we insert a directed edge from $s$ to each vertex in $X$, and a directed edge from each vertex in $Y$ to $t$. Finally, we assign to each edge of $H$ a capacity of 1.

- Given a flow f for H, we use f to define a set M of edges of G using the rule that an edge e is in M whenever f(e) = 1.

# Example and Analysis

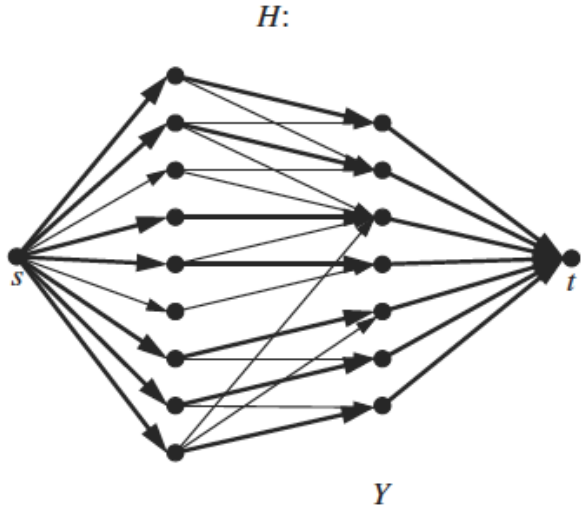



**Figure 16.11:** (a) A bipartite graph $G$. (b) Flow network $H$ derived from $G$ and a maximum flow in $H$; thick edges have unit flow and other edges have zero flow.

- Running time is O(nm), because G is connected.

# Baseball Elimination

- Let T be a set of teams in a sports league, which, for historical reasons, let us assume is baseball.
- At any point during the season, each team, i, in T, will have some number, $w_i$, of wins, and will have some number, $g_i$, of games left to play.
- The **baseball elimination problem** is to determine whether it is possible for team i to finish the season in first place, given the games it has already won and the games it has left to play.
- Note that this depends on more than just the number of games left for team i, however; it also depends on the respective schedules of team i and the other teams.

# Baseball Elimination Example

- Let $g_{i,j}$ denote the number of games remaining between team i and team j, so that $g_i$ is the sum, over all j, of the $g_{i,j}$'s.

| Team $i$ | Wins $w_i$ | Games Left $g_i$ | Schedule $(g_{i,j})$ LA | Oak | Sea | Tex |
|---|---|---|---|---|---|---|
| Los Angeles | 81 | 8 | - | 1 | 6 | 1 |
| Oakland | 77 | 4 | 1 | - | 0 | 3 |
| Seattle | 76 | 7 | 6 | 0 | - | 1 |
| Texas | 74 | 5 | 1 | 3 | 1 | - |

**Table 16.12:** A set of teams, their standings, and their remaining schedule. Clearly, Texas is eliminated from finishing in first place, since it can win at most 79 games. In addition, even though it is currently in second place, Oakland is also eliminated, because it can win at most 81 games, but in the remaining games between LA and Seattle, either LA wins at least 1 game and finishes with at least 82 wins or Seattle wins 6 games and finishes with at least 82 wins.

# Reduction to Max Flow

With all the different ways for a team, $k$, to be eliminated, it might at first seem like it is computationally infeasible to determine whether team $k$ is eliminated. Still, we can solve this problem by a reduction to a network flow problem. Let $T'$ denote the set of teams other than $k$, that is, $T' = T - \{k\}$. Also, let $L$ denote the set of games that are left to play among teams in $T'$, that is,

$$L = \{\{i, j\} : \ i, j \in T' \text{ and } g_{i,j} > 0\}.$$

Finally, let $W$ denote the largest number of wins that are possible for team $k$ given the current standings, that is, $W = w_k + g_k$.

- Let us assume no single team eliminates team k (since this is easy to check).

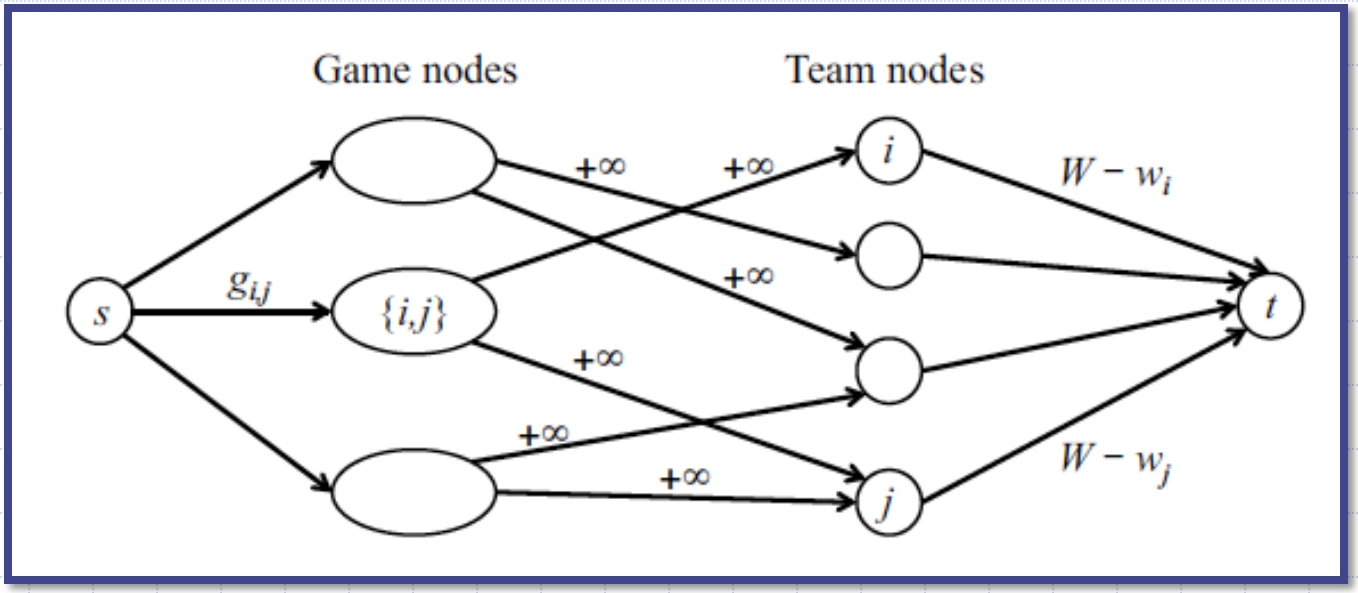
# Creating the Graph

- To consider how a combination of teams and game outcomes might eliminate team k, we create a graph, G, that has as its vertices a source, s, a sink, t, and the sets T′ and L. Then, let us include the following edges in G:

- For each game pair, $\{i, j\}$, in $L$, add an edge $(s, \{i, j\})$, and give it capacity $g_{i,j}$.
- For each game pair, $\{i, j\}$, in $L$, add edges $(\{i, j\}, i)$ and $(\{i, j\}, j)$, and give these edges capacity $+\infty$.
- For each team, $i$, add an edge $(i, t)$ and give it capacity $W - w_i$, which cannot be negative in this case, since we ruled out the case when $W < w_i$.

# Creating the Graph, Example

- For each game pair, $\{i, j\}$, in $L$, add an edge $(s, \{i, j\})$, and give it capacity $g_{i,j}$.
- For each game pair, $\{i, j\}$, in $L$, add edges $(\{i, j\}, i)$ and $(\{i, j\}, j)$, and give these edges capacity $+\infty$.
- For each team, $i$, add an edge $(i, t)$ and give it capacity $W - w_i$, which cannot be negative in this case, since we ruled out the case when $W < w_i$.



# Intuition and Analysis

The intuition behind the construction for $G$ is that wins flow out from the source, $s$, are split at each game node, $\{i, j\}$, to allocate wins between each pair of teams, $i$ and $j$, and then are absorbed by the sink, $t$. The flow on each edge, $(\{i, j\}, i)$, represents the number of games in which team $i$ beats $j$, and the flow on each edge, $(i, t)$, represents the number of remaining games that could be won by team $i$. Thus, maximizing the flow in $G$ is equivalent to testing if it is possible to allocate wins among all the remaining games not involving team $k$ so that no team goes above $W$ wins. So we compute a maximum flow for $G$.

- We can solve baseball elimination for any team in a set of n teams by solving a single maximum flow problem on a network with at most $O(n^2)$ vertices and edges.