COL106: Data Structures and Algorithms

Ragesh Jaiswal, IIT Delhi

Network Flow

- Suppose there are four teams in IPL with their current number of wins:
  - Daredevils: 10
  - Sunrisers: 10
  - Lions: 10
  - Supergiants: 8
- There are 7 more games to be played. These are as follows:
  - Supergiants plays all other 3 teams.
  - Daredevils Vs Sunrisers, Sunrisers Vs Lions, Daredevils Vs Lions, Sunrisers Vs Daredevils

- Suppose there are four teams in IPL with their current number of wins:
  - Daredevils: 10
  - Sunrisers: 10
  - Lions: 10
  - Supergiants: 8
- There are 7 more games to be played. These are as follows:
  - Supergiants plays all other 3 teams.
  - Daredevils Vs Sunrisers, Sunrisers Vs Lions, Daredevils Vs Lions, Sunrisers Vs Daredevils
- A team is said to be eliminated if it cannot end with maximum number of wins.
- Can we say that Supergiants have been eliminated give the current scenario?

- Suppose there are four teams in IPL with their current number of wins:
  - Daredevils: 10
  - Sunrisers: 10
  - Lions: 9
  - Supergiants: 8
- There are 7 more games to be played. These are as follows:
  - Supergiants plays all other 3 teams.
  - 4 games between Daredevils and Sunrisers.
- Can we say that Supergiants have been eliminated give the current scenario?

Problem

There are n teams. Each team i has a current number of wins denoted by $w(i)$.There are $G(i,j)$ games yet to be played between team i and j. Design an algorithm to determine whether a given team x has been eliminated.

Problem

There are n teams. Each team i has a current number of wins denoted by $w(i)$.There are $G(i,j)$ games yet to be played between team i and j. Design an algorithm to determine whether a given team x has been eliminated.

- Consider the following flow network

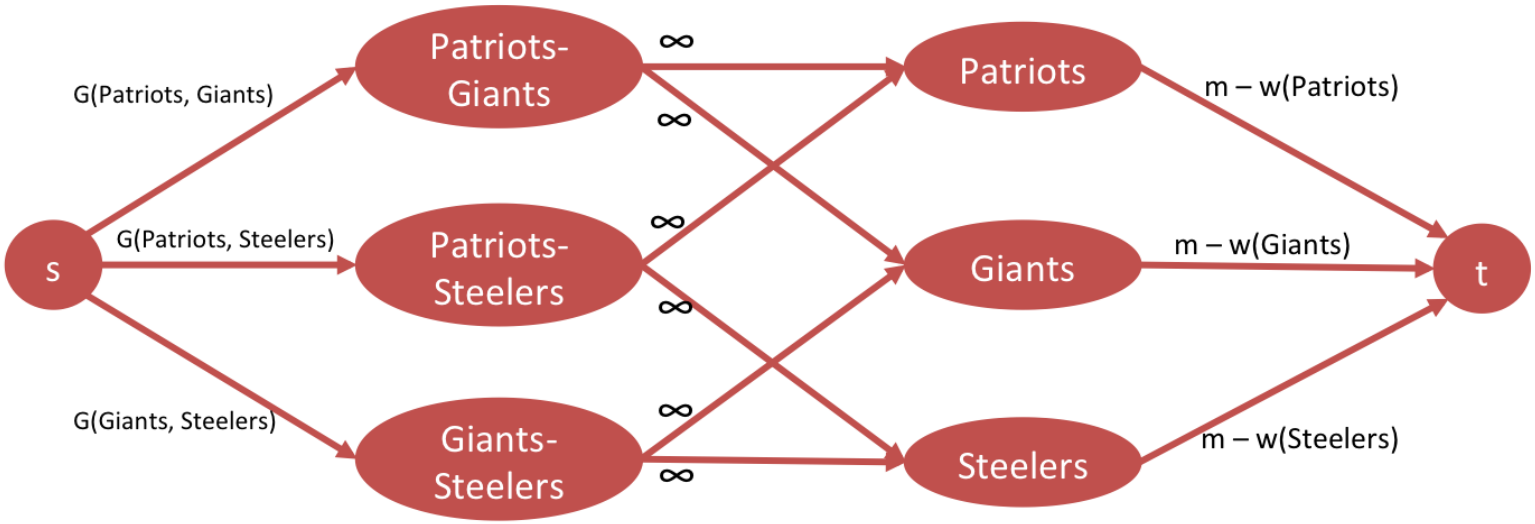


Figure: Team x can end with at most m wins, i.e., $m = w(x) + \sum_j G(x,j)$

Problem

There are n teams. Each team i has a current number of wins denoted by $w(i)$.There are $G(i,j)$ games yet to be played between team i and j. Design an algorithm to determine whether a given team x has been eliminated.

- Claim 1: Team x has been eliminated **iff** the maximum flow in the network is $< g^*$, where $g^* = \sum_{i,j \text{ s.t. } x \notin \{i,j\}} G(i,j)$.

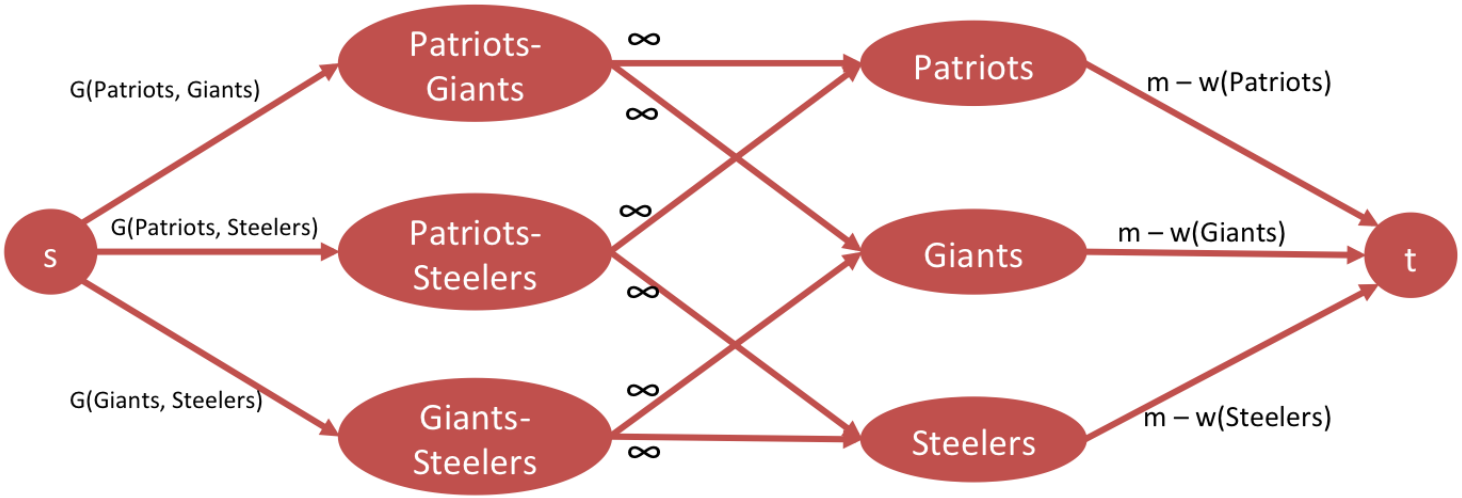
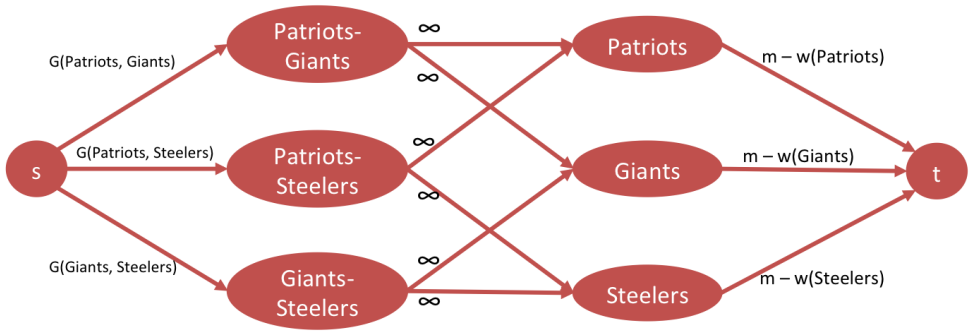


Figure: Team x can end with at most m wins, i.e., $m = w(x) + \sum_j G(x,j)$

Problem

There are n teams. Each team i has a current number of wins denoted by $w(i)$.There are $G(i,j)$ games yet to be played between team i and j. Design an algorithm to determine whether a given team x has been eliminated.

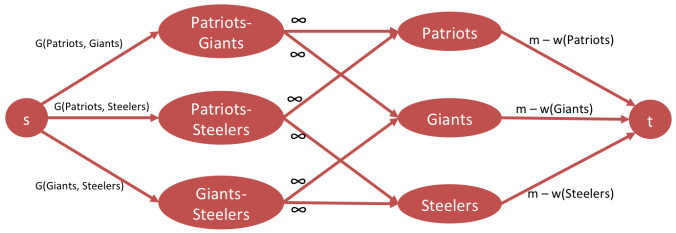
- Claim 1: Team x has been eliminated **iff** the maximum flow in the network is $< g^*$, where $g^* = \sum_{i,j \text{ s.t. } x \notin \{i,j\}} G(i,j)$.
- Comment: If we can somehow find a subset T of teams (not including x) such that $\sum_{i \in T} w(i) + \sum_{i<j \text{ and } i,j \in T} G(i,j) > m \cdot |T|$. Then we have a witness to the fact that x has been eliminated.



Problem

There are n teams. Each team i has a current number of wins denoted by $w(i)$.There are $G(i,j)$ games yet to be played between team i and j. Design an algorithm to determine whether a given team x has been eliminated.

- Claim 1: Team x has been eliminated **iff** the maximum flow in the network is $< g^*$, where $g^* = \sum_{i,j \text{ s.t. } x \notin \{i,j\}} G(i,j)$.
- Comment: If we can somehow find a subset T of teams (not including x) such that $\sum_{i \in T} w(i) + \sum_{i<j \text{ and } i,j \in T} G(i,j) > m \cdot |T|$. Then we have a witness to the fact that x has been eliminated.
- Can we find such a subset T?

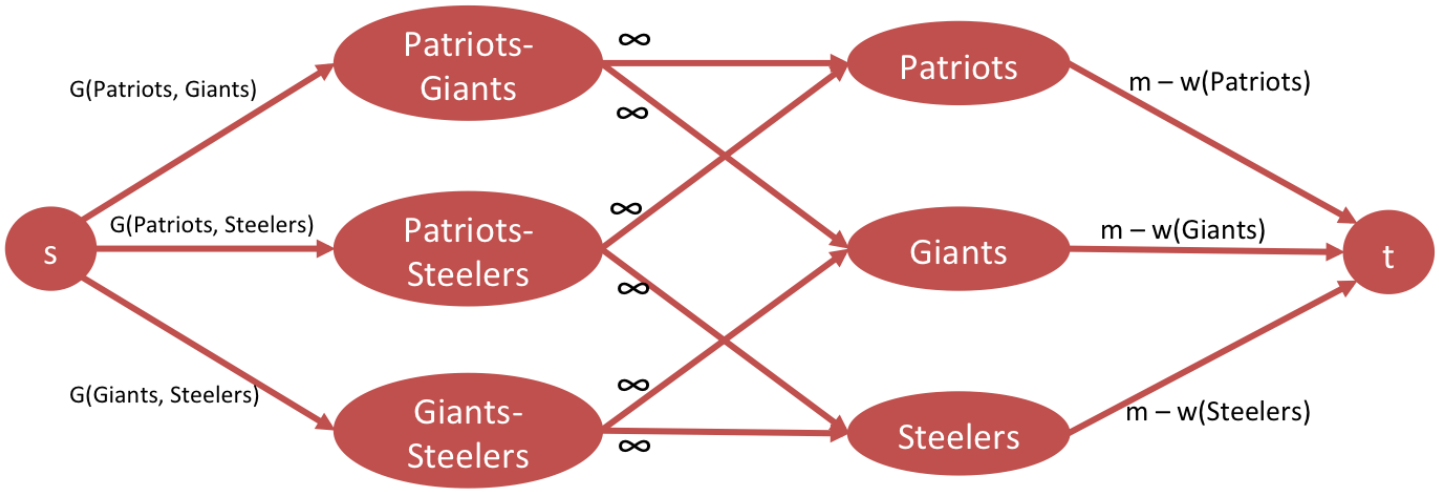


- Claim 1: Team x has been eliminated **iff** the maximum flow in the network is $< g^*$, where $g^* = \sum_{i,j \text{ s.t. } x \notin \{i,j\}} G(i,j)$.

Proof.

- Claim 1.1: If x has been eliminated, then the max flow in the network is $< g^*$.

□



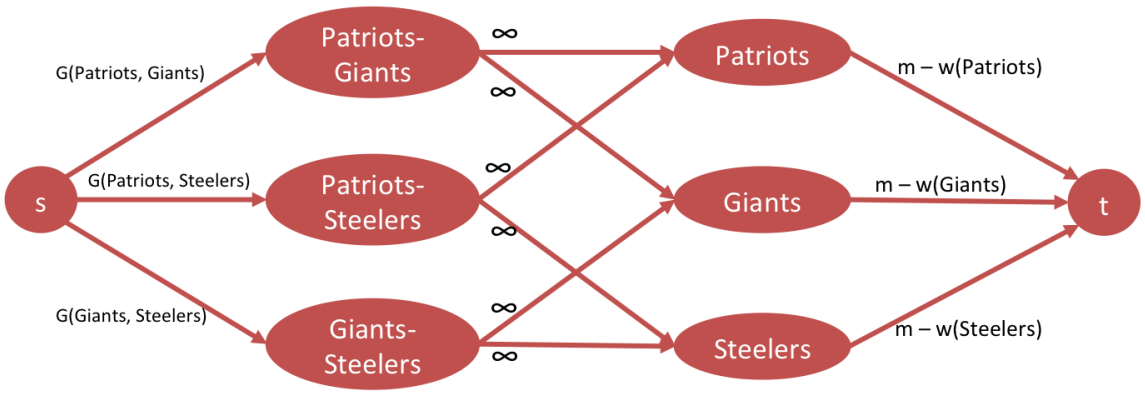
- Claim 1: Team x has been eliminated **iff** the maximum flow in the network is $< g^*$, where $g^* = \sum_{i,j \text{ s.t. } x \notin \{i,j\}} G(i,j)$.

Proof of Claim 1

- Claim 1.1: If x has been eliminated, then the max flow in the network is $< g^*$.
- Claim 1.2: If the max flow is $< g^*$, then team x has been eliminated.

Proof of Claim 1.2

- Consider any s-t min-cut (A, B) in the graph.
- Claim 1.2.1: If v_{ij} is in A, then both v_i and v_j are in A.



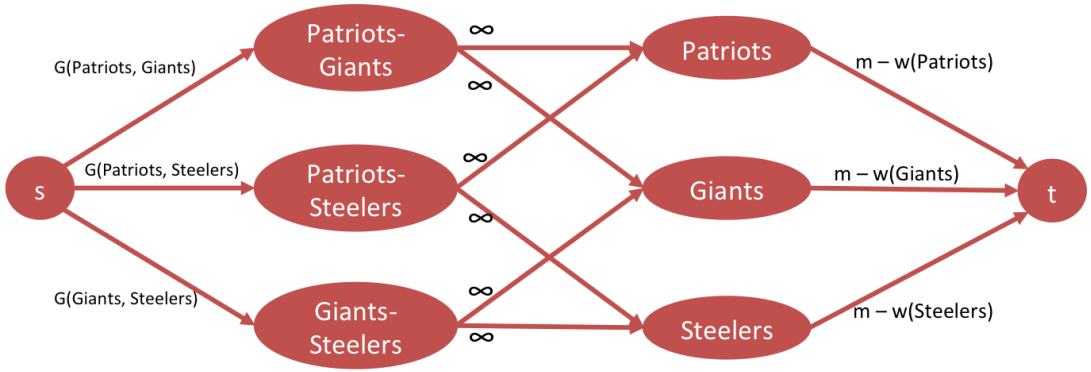
- Claim 1: Team x has been eliminated **iff** the maximum flow in the network is $< g^*$, where $g^* = \sum_{i,j \text{ s.t. } x \notin \{i,j\}} G(i,j)$.

Proof of Claim 1

- Claim 1.1: If x has been eliminated, then the max flow in the network is $< g^*$.
- Claim 1.2: If the max flow is $< g^*$, then team x has been eliminated.

Proof of Claim 1.2

- Consider any s-t min-cut (A, B) in the graph.
- Claim 1.2.1: If v_{ij} is in A, then both v_i and v_j are in A.
- Claim 1.2.2: If both v_i and v_j are in A, then v_{ij} is in A.



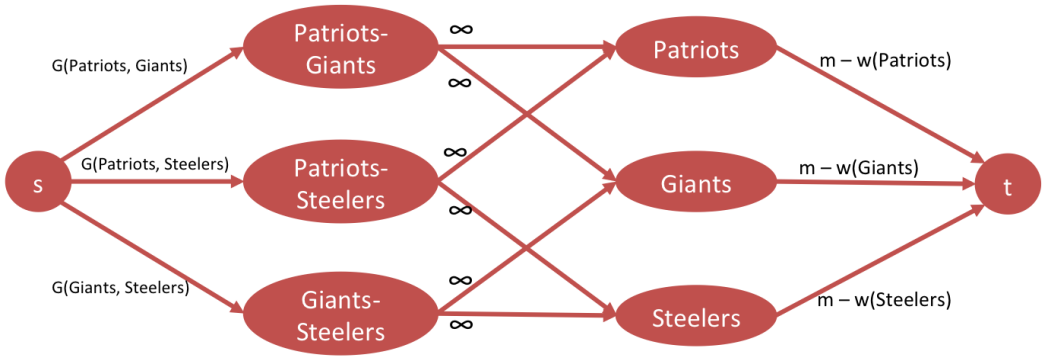
- Claim 1: Team x has been eliminated **iff** the maximum flow in the network is $< g^*$, where $g^* = \sum_{i,j \text{ s.t. } x \notin \{i,j\}} G(i,j)$.

Proof of Claim 1

- Claim 1.1: If x has been eliminated, then the max flow in the network is $< g^*$.
- Claim 1.2: If the max flow is $< g^*$, then team x has been eliminated.

Proof of Claim 1.2

- Consider any s-t min-cut (A, B) in the graph.
- Claim 1.2.1: If v_{ij} is in A, then both v_i and v_j are in A.
- Claim 1.2.2: If both v_i and v_j are in A, then v_{ij} is in A.
- Let T be the set of teams such that $i \in T$ **iff** $v_i \in A$.



- Claim 1: Team x has been eliminated **iff** the maximum flow in the network is $< g^*$, where $g^* = \sum_{i,j \text{ s.t. } x \notin \{i,j\}} G(i,j)$.

Proof of Claim 1

- Claim 1.1: If x has been eliminated, then the max flow in the network is $< g^*$.
- Claim 1.2: If the max flow is $< g^*$, then team x has been eliminated.

Proof of Claim 1.2

- Consider any s-t min-cut (A, B) in the graph.
- Claim 1.2.1: If v_{ij} is in A, then both v_i and v_j are in A.
- Claim 1.2.2: If both v_i and v_j are in A, then v_{ij} is in A.
- Let T be the set of teams such that $i \in T$ **iff** $v_i \in A$. Then we have:

$$C(A,B) = \sum_{i \in T}(m - w(i)) + \sum_{\{i,j\} \not\subset T} G(i,j) < g^*$$

$$\Rightarrow \quad m \cdot |T| - \sum_{i \in T} w(i) + (g^* - \sum_{\{i,j\} \subset T} G(i,j)) < g^*$$

$$\Rightarrow \quad \sum_{i \in T} w(i) + \sum_{\{i,j\} \subset T} G(i,j) > m \cdot |T| \quad \square$$

$\square$

Course Overview

- Basic graph algorithms
- Algorithm Design Techniques:
  - Greedy Algorithms
  - Divide and Conquer
  - Dynamic Programming
  - Network Flow
- Computational Intractability

Computational Intractability

Definition (Efficient Algorithms)

An algorithm is said to be *efficient* iff it runs in time polynomial in the input size. Such algorithms are also called *polynomial-time* algorithms.

Definition (Efficient Algorithms)

An algorithm is said to be *efficient* iff it runs in time polynomial in the input size. Such algorithms are also called *polynomial-time* algorithms.

- Question 1: Given a problem, does there exist an efficient algorithm to solve the problem?

Definition (Efficient Algorithms)

An algorithm is said to be *efficient* iff it runs in time polynomial in the input size. Such algorithms are also called *polynomial-time* algorithms.

- Question 1: Given a problem, does there exist an efficient algorithm to solve the problem?
- There are lots of problems arising in various fields for which this question is unresolved.
- Question 2: Are these problems related in some manner?

Definition (Efficient Algorithms)

An algorithm is said to be *efficient* iff it runs in time polynomial in the input size. Such algorithms are also called *polynomial-time* algorithms.

- Question 1: Given a problem, does there exist an efficient algorithm to solve the problem?
- There are lots of problems arising in various fields for which this question is unresolved.
- Question 2: Are these problems related in some manner?
- Question 3: If someone discovers an efficient algorithm to one of these difficult problems, then does that mean that there are efficient algorithms for other problems? If so, how do we obtain such an algorithm.

- NP-complete problems: This is a large class of problems such that all problems in this class are equivalent in the following sense:

  The existence of a polynomial-time algorithm for any one problem in this class implies the existence of polynomial-time algorithm for all of them.

- NP-complete problems: This is a large class of problems such that all problems in this class are equivalent in the following sense:

  The existence of a polynomial-time algorithm for any one problem in this class implies the existence of polynomial-time algorithm for all of them.

- Polynomial-time reduction:
  - Consider two problems X and Y.
  - Suppose there is a *black box* that solves arbitrary instances of problem X.
  - Suppose any arbitrary instance of problem Y can be solved using a polynomial number of standard computational steps and a polynomial number of calls to the black box that solves instance of problem X.
  - If the previous statement is true, then we say that Y is polynomial-time reducible to X. A short notation for this is $Y \leq_p X$.

- Polynomial-time reduction:
  - Consider two problems X and Y.
  - Suppose there is a *black box* that solves arbitrary instances of problem X.
  - Suppose any arbitrary instance of problem Y can be solved using a polynomial number of standard computational steps and a polynomial number of calls to the black box that solves instance of problem X.
  - If the previous statement is true, then we say that Y is polynomial-time reducible to X. A short notation for this is $Y \leq_p X$.
- Claim 1: BIPARTITE-MATCHING $\leq_p$ MAX-FLOW.

- Polynomial-time reduction:
  - Consider two problems X and Y.
  - Suppose there is a *black box* that solves arbitrary instances of problem X.
  - Suppose any arbitrary instance of problem Y can be solved using a polynomial number of standard computational steps and a polynomial number of calls to the black box that solves instance of problem X.
  - If the previous statement is true, then we say that Y is polynomial-time reducible to X. A short notation for this is $Y \leq_p X$.
- Claim 2: Suppose $Y \leq_p X$. If X can be solved in polynomial time, then Y can be solved in polynomial time.

- Polynomial-time reduction:
  - Consider two problems X and Y.
  - Suppose there is a *black box* that solves arbitrary instances of problem X.
  - Suppose any arbitrary instance of problem Y can be solved using a polynomial number of standard computational steps and a polynomial number of calls to the black box that solves instance of problem X.
  - If the previous statement is true, then we say that Y is polynomial-time reducible to X. A short notation for this is $Y \leq_p X$.
- Claim 2: Suppose $Y \leq_p X$. If X can be solved in polynomial time, then Y can be solved in polynomial time.
- Claim 3: Suppose $Y \leq_p X$. If Y cannot be solved in polynomial time, then X cannot be solved in polynomial time.

End