CSE 417: Algorithms and Computational Complexity

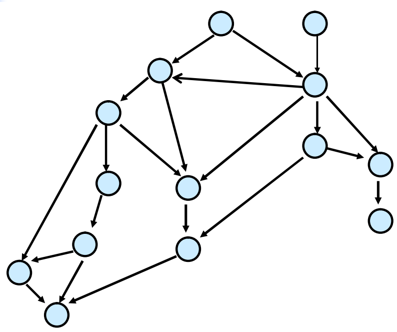
More Graph Algorithms

Autumn 2002
Paul Beame

Topological Sort

- **Given:** a directed acyclic graph (DAG) **G**=(**V**,**E**)
- **Output:** numbering of the vertices of **G** with distinct numbers from **1** to **n** so edges only go from lower number to higher numbered vertices
- Applications
  - nodes represent tasks
  - edges represent precedence between tasks
  - topological sort gives a sequential schedule for solving them

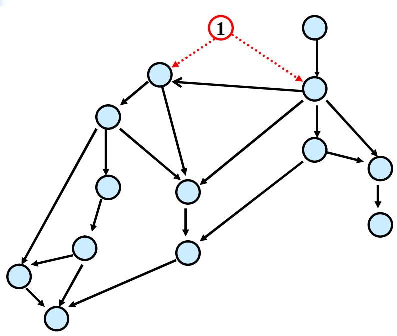
Directed Acyclic Graph



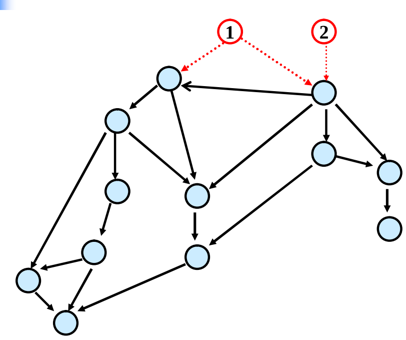
Topological Sort

- Can do using DFS (see book)
- Alternative simpler idea:
  - Any vertex of in-degree 0 can be given number 1 to start
  - Remove it from the graph and then give a vertex of in-degree 0 number 2, etc.

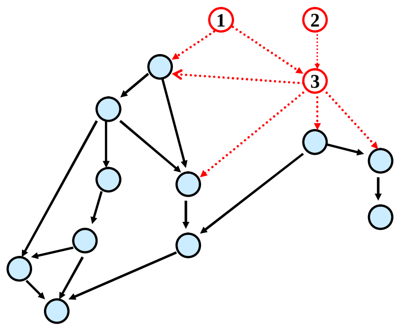
Topological Sort



Topological Sort

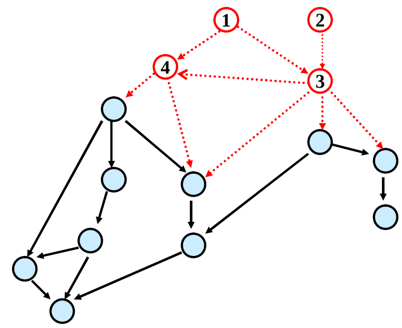


Topological Sort



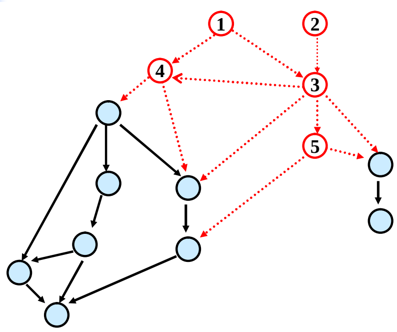
7

Topological Sort



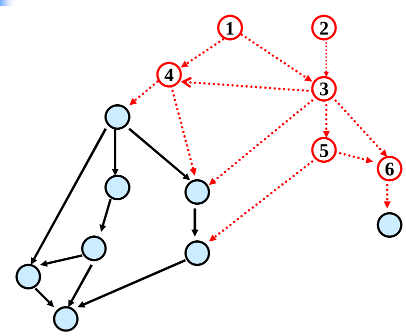
8

Topological Sort



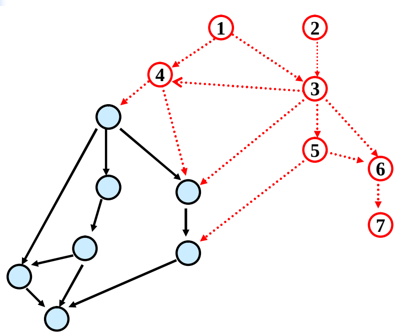
9

Topological Sort



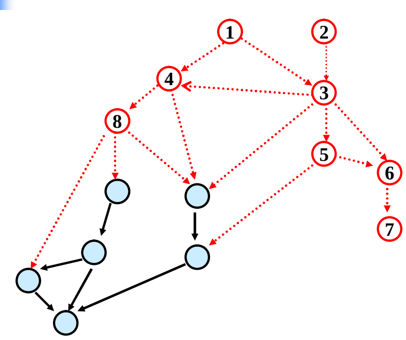
10

Topological Sort



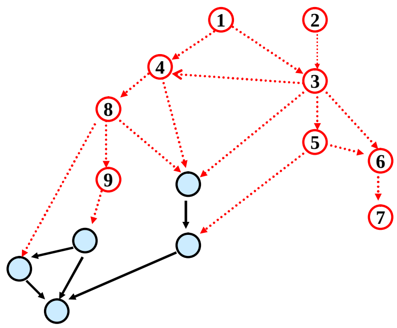
11

Topological Sort



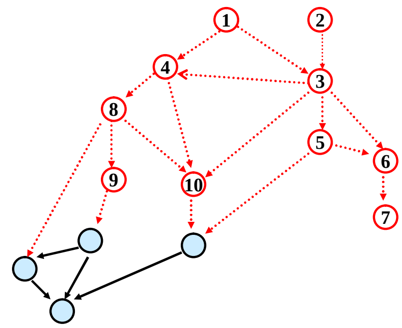
12

Topological Sort



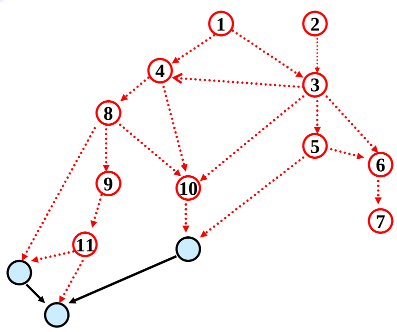
13

Topological Sort



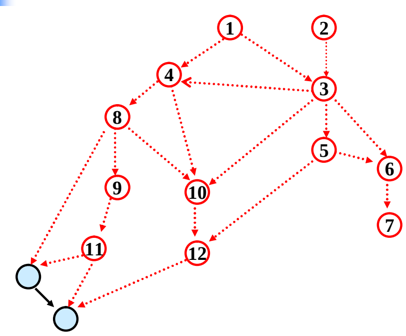
14

Topological Sort



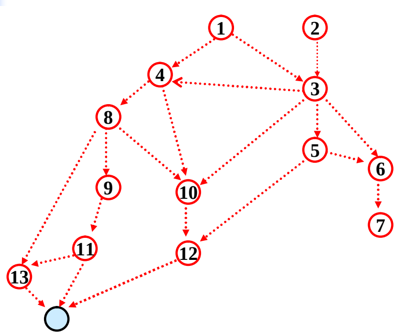
15

Topological Sort



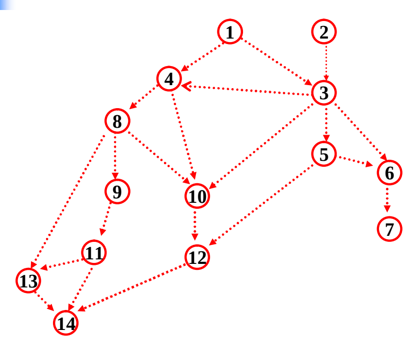
16

Topological Sort



17

Topological Sort



18

Implementing Topological Sort

- Go through all edges, computing in-degree for each vertex **O(m+n)**
- Maintain a queue (or stack) of vertices of in-degree **0**
- Remove any vertex in queue and number it
- When a vertex is removed, decrease in-degree of each of its neighbors by **1** and add them to the queue if their degree drops to **0**
- Total cost **O(m+n)**

Matchings and Bipartite Graphs

- Given a graph **G=(V,E)**
  - a subset **M** Í **E** of edges is a **matching** iff no two edges of **M** share an endpoint
- Many graphs in practice have two types of nodes and all edges join nodes of different types
  - e.g., each node is either
    - the name **i** of an instructor or
    - a course number **c**

    (i,c) is an edge iff instructor **i** can teach course **c**
  - These graphs are called **bipartite**

The Maximum Matching Problem

- Often would like to find a **perfect** matching – one with an edge touching every node in the graph
  - In our example, a perfect matching would correspond to an assignment of instructors to courses that would make sure that every course is covered and every instructor is busy teaching
- More generally, the **maximum matching problem** asks the following
  - Given: a graph **G=(V,E)**
  - Find: a matching **M** in **G** such that contains as many edges as possible

A Greedy Approach

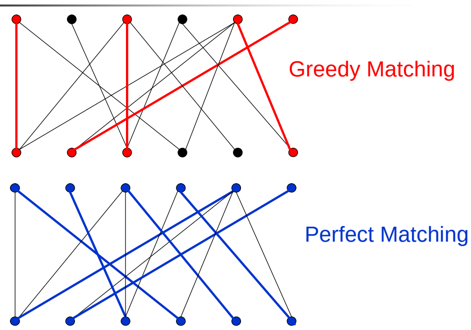
M? Æ

while (there is some edge **e**∈**E**

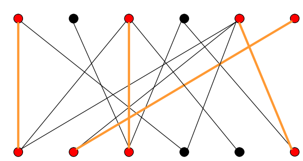
not touching any edge of **M**) do

Add **e** to **M**

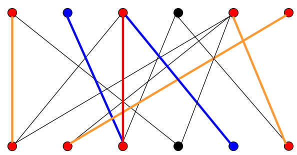
Greed doesn’t always work



Start with a Greedy Matching



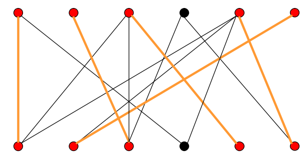
Improving a Matching



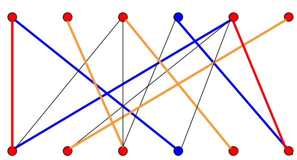
Replace the **red edge** with the **two blue edges**

Based on a path that **starts** and **ends** at an unmatched vertex

Improving a Matching



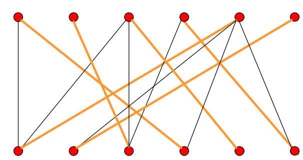
Improving a Matching



Replace the **red edges** with the **blue edges**

Based on a path that **starts** and **ends** at an unmatched vertex

Improving a Matching



Alternating Paths

- Given a graph **G** and a matching **M** in **G**, an **alternating path** is a path that
  - Starts at an unmatched vertex of **G**
  - Ends at an unmatched vertex of **G**
  - Has edges that alternate between being in **M** and not being in **M**
- If there is an alternating path **P** in graph **G** with respect to **M** then we can improve **M** by "flipping" edges on the path, i.e.
  - Remove all edges of **P** previously in **M**
  - Add all edges of **P** previously not in **M**

Alternating Paths

- Maximum Bipartite Matching Algorithm:

  M ¬ greedy matching
  while (there is an alternating path **P** with
  respect to **M**) do
  flip the edges along **P**

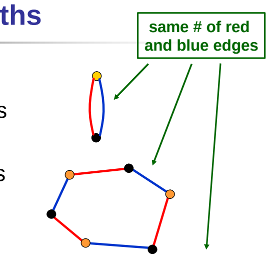
- Why does it work?
  - Need to show that
    - if a larger matching than **M** exists in **G** then there will be an alternating path in **G** with respect to M

Alternating paths exist

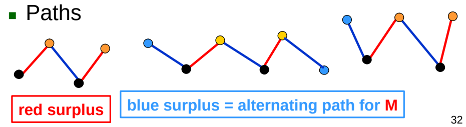
- Lemma: Suppose that **M** and **M'** are matchings in graph **G** and **|M|<|M'|** then there is an alternating path in **G** with respect to **M**
- Proof idea:
  - Take the graph consisting of the edges of both **M** and **M'**
  - Every vertex is touched by at most 2 edges
    - Graph consists of a collection of paths and cycles

Alternating Paths

- Possibilities
  - Overlapping edges
  - Even length cycles



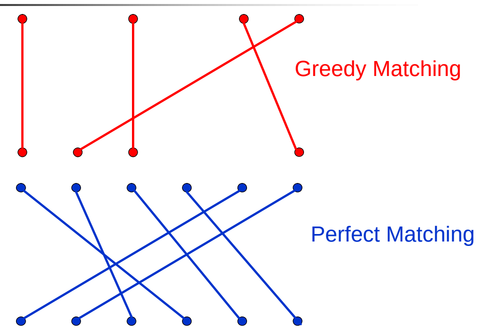
  - Paths



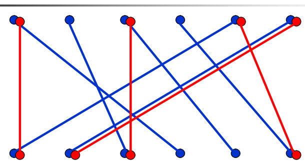
Alternating Paths

- **M'** has more edges than M does
  - the graph of **M∪M'** must have a component with a blue surplus
  - Therefore **G** has an alternating path with respect to **M**

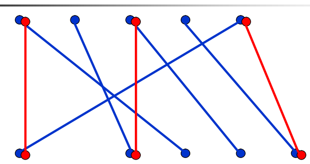
Our example



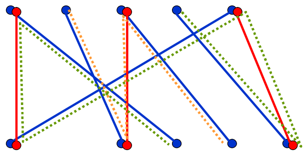
Our example



Ignore places where M and M' agree

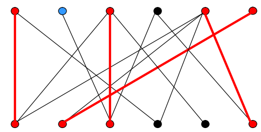


Alternating Paths



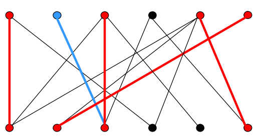
37

Finding Alternating Paths



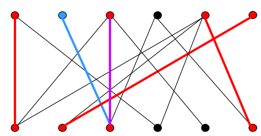
38

Finding Alternating Paths



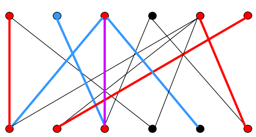
39

Finding Alternating Paths



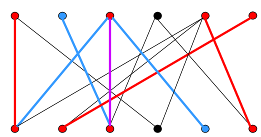
40

Finding Alternating Paths



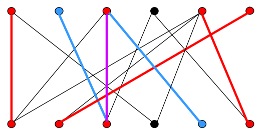
41

Finding Alternating Paths



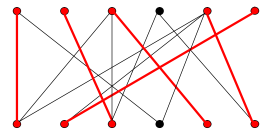
42

Finding Alternating Paths



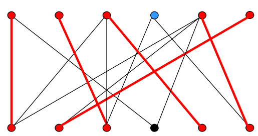
43

Finding Alternating Paths



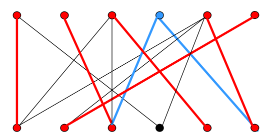
44

Finding Alternating Paths



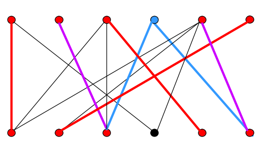
45

Finding Alternating Paths



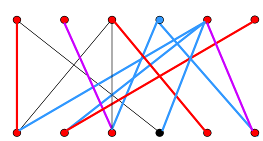
46

Finding Alternating Paths



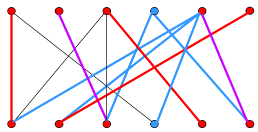
47

Finding Alternating Paths

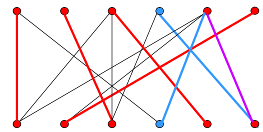


48

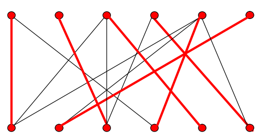
Finding Alternating Paths



Finding Alternating Paths



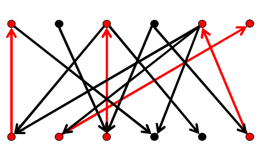
Finding Alternating Paths



Finding Alternating Paths

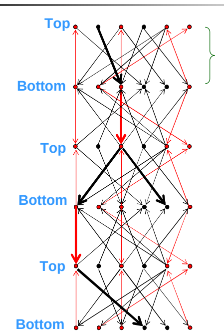
- The search was like breadth-first search
  - except that when we hit a matched edge we were forced to follow it
- We traversed
  - unmatched edges from top to bottom
  - matched edges from bottom to top
- To enforce this behavior
  - Direct all unmatched edges top to bottom
  - Direct all matched edges bottom to top

Directing the graph



Now run ordinary breadth-first search
from each unmatched node on top
until we reach another unmatched node
(which will be on the bottom)

Searching from a single unmatched node



Original graph

In this picture the graph is repeated so it is easier to see the execution

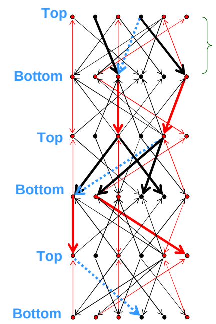
The actual search works on the original graph

Running time for matching

- Finding the greedy matching is **O(n+m)** time
- Finding each alternating path is BFS
  - **O(n+m)** time
- Each alternating path increases matching size by **1**
  - Total of at most **n/2** rounds of finding alternating paths
- Total run time **O(nm+n²)**

- Can do a bit better

Searching from all top unmatched nodes in one round of BFS

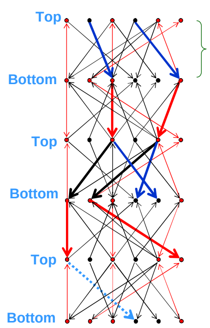


Original graph

In this picture the graph is repeated so it is easier to see the execution

The actual search works on the original graph

Searching from all top unmatched nodes in one round of BFS



Original graph

Flip more than one alternating path at the same time

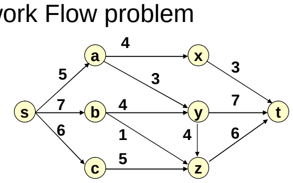
Since each node appears at most once in a BFS tree these paths don't conflict

BFS from multiple unmatched nodes

- If the algorithm
  - does a single BFS from all unmatched top nodes
  - stops at the level where the first unmatched bottom node is found
  - flips **all** alternating paths that reach that level
- Then
  - Only $O(\sqrt{n})$ rounds needed (proof is complicated)
    - Total $O(m\sqrt{n} + n^{3/2})$ time needed

Using similar ideas can solve

- Network Flow problem



  - How much stuff can flow from s to t?
- Lots of applications

Bipartite matching as a special case of flow