CS 106B, Lecture 23
The Power of DFS
Plan for Today

• Two different graph algorithms
  – Topological Sort
  – Bipartite Graph Matching
• Modify DFS for powerful results
Recap: Depth-First Search

- Path-finding algorithm
- Pseudocode:
  
  ```
  dfs from \( v_1 \):
  
  mark \( v_1 \) as seen.
  
  for each of \( v_1 \)'s unvisited neighbors \( n \):
    dfs\((n)\)
  ```

- Can also run depth-first searching looking for a **specific** endpoint
  - Check out the "find all solutions" vs. "find one solution" pseudocode from recursive backtracking
A new problem

• In what order can you take the CS classes required for the major?
  – Some classes rely on other classes – you shouldn't take 106B until you've taken 106A

• Another example: you want to cook breakfast, but some steps must be done before others can begin. In what order should you perform the steps to cook breakfast?

• In what order should compilers compile code (with import statements)?

• What type of graphs are these?
Topological Sort

• Want to order tasks such that every task's prerequisites appear before the task itself

• In other words, if 106A is a prerequisite for 106B, 106A should be before 106B in the ordering

• Such an ordering is a **topological ordering** and is created using **topological sort**

• Only works on **directed, acyclic graphs**
  – Prerequisite relationships are always directed
  – If the graph has cycles, no way to obey all the prerequisites
Topological Ordering

- wash veggies
- get butter
- sauté veggies
- make omelet
- get bread
- butter toast
- get utensils
- eat!
Topological Ordering

- Any of the top four tasks can be done in any order (no prerequisites)

- Tasks:
  - Wash veggies
  - Sauté veggies
  - Get butter
  - Toast bread
  - Get utensils
  - Make omelet
  - Eat!
Topological Ordering

- **Butter toast**'s prerequisites have all been met, so can do that next.
Topological Ordering

- Can sauté the vegetables since we've already washed the veggies and gotten butter

Diagram:
1. Wash veggies
2. Get butter
3. Toast bread
4. Get utensils
5. Make omelet
6. Eat!
Topological Ordering

- Can make the omelet
• Finally, we can eat!
• This is just one topological ordering – what's another?
A Note About DFS

• In what order do we finish visiting nodes (do they turn grey in our example from Thursday) in DFS on a DAG?
DFS on a DAG

1. Wash veggies
2. Get butter
3. Toast bread
4. Get utensils
5. Sauté veggies
6. Make omelet
7. Eat!
Topological Sort with DFS

• Key observation: finishing visiting node $a$ means we must have visited all nodes that have $a$ as a prerequisite

• How could we modify DFS to return the topological ordering?
  – We'll need a Vector to maintain the order we traverse nodes
  – In what order should we add the nodes to the Vector? Where should we add the node (beginning/random place/end)?
Topological Sort Algorithm

For each unvisited node:
   run TopoDFS(node)

TopoDFS(node):
   if we've seen this node before while running DFS, there's a cycle!
   run TopoDFS on each of the node's neighbors
   add node to the front of the ordering
   node is now visited
• You should be working on Autocomplete
• Please give us feedback! cs198.stanford.edu
• Feel free to use seepluspl.us to help you understand trees or pointers. It's still in development, so be patient with quirks
• Course feedback:
  – You all like that I write code in class – we'll get back to doing that by the end of this week
  – It's a hard class, but you all are doing fantastically
    • Please ask questions on Piazza, come talk to me after class, email me for a meeting, etc. if you feel like you're falling behind or don't understand the material
  – We've set grading deadlines before each assignment is due – if you haven't received a grade from your SL by the time the next assignment is due, email them (we also tell them)
Another Type of Graph

• Sometimes, we want to model problems like assigning:
  – Doctors and patients
  – Students and classes
  – Classes and rooms

• Key properties:
  – we have two different types of nodes
  – all the relationships (edges) are between nodes of different types
    • e.g. a student is assigned to a class – no relationships between students or between classes

• A **bipartite graph** is a graph with two types of nodes (left-hand side and right-hand side), where all the (undirected) edges go from the LHS to the RHS
Since Dr. A is both a patient and a doctor, this is not a bipartite graph.
Bipartite Graph Matching

- A **matching** is a set of edges such that each node is connected to at most one edge
  - **Maximum matching**: largest such set of edges
Matching Algorithm

• Start with an empty matching

• For each LHS node, either:
  – Match it to an unmatched RHS neighbor
  – Match it to a matched RHS neighbor and break the RHS neighbor's match, then try to match the newly unmatched LHS node. If you can't, keep the old matching

• How is this algorithm like depth-first search?
Bipartite Graph Algorithm
Bipartite Graph Algorithm
Bipartite Graph Algorithm
Bipartite Graph Algorithm
Bipartite Graph Algorithm
Bipartite Graph Algorithm

Better match found!
Bipartite Graph Algorithm
No better match found!
Bipartite Graph Algorithm
Bipartite Graph Algorithm

No better match found!
Bipartite Graph Algorithm
Bipartite Graph Algorithm
Bipartite Graph Algorithm

No better match found!
Bipartite Graph Algorithm
Bipartite Graph Algorithm

Maximum matching
Matching Algorithm

• Start with an empty matching

• For each LHS node, either:
  – Match it to an unmatched RHS neighbor
  – Match it to a matched RHS neighbor and break the RHS neighbor's
    match, then try to match the newly unmatched LHS node. If you can't,
    keep the old matching
An observation

• Breaking an already made match and finding a better match means an **alternating path** from an unmatched LHS node to an unmatched RHS node
An observation

- Bigger example:
An observation

• Bigger example:
An observation

• Bigger example:
An observation

• Bigger example:
An observation

- Bigger example:
An observation

- Notice how we augmented the alternating path by adding two new nodes (an unmatched LHS node and an unmatched RHS node)
  - The previous matching is now red, excluded from current matching
An observation

• The **black** edges are in the matching, and the **red** edges are not
  – **black** is LHS to RHS, **red** is RHS to LHS
Alternate Approach

• Start with an empty matching

• While possible:
  – Find an alternating path from an unmatched LHS node to an unmatched RHS, potentially by augmenting an existing alternating path
    • The black edges in such a path (from LHS to RHS) are included in the matching; the red edges (from RHS to LHS) are not
  – If no such path exists, we've found the maximum matching

• How do we find a path?