CS 491 CAP
Introduction to Graphs and Search

Jingbo Shang

University of Illinois at Urbana-Champaign

Sep 15, 2017
Outline

◊ Graphs
◊ Adjacency Matrix vs. Adjacency List
◊ Special Graphs
◊ Depth-first and Breadth-first Search
◊ Topological Sort
Outline

- **Graphs**
- Adjacency Matrix vs. Adjacency List
- Special Graphs
- Depth-first and Breadth-first Search
- Topological Sort
Graphs

- Graph is an abstract way of representing connectivity using nodes (vertices) and edges (arcs)
- \( n \) nodes, labeled from 1 to \( n \)
- \( m \) edges connect some pairs of nodes
  - either directed (unidirected) or bidirectional (undirected)
- Nodes and edges can carry some extra information such as weights
Graph Problems

- Shortest path
- Minimum spanning tree
- Matching / Network flow
- 2-SAT
- Graph coloring
- Traveling salesman problem
- ...

CS 491 CAP – Intro to Competitive Algorithmic Programming
Outline

- Graphs
- **Adjacency Matrix vs. Adjacency List**
- Special Graphs
- Depth-first and Breadth-first Search
- Topological Sort
Graph storage

◊ Adjacency matrix
  - bool mat[n][n]
  - mat[u][v] is the indicator that whether there is an edge between node u and v.

◊ Adjacency list
  - vector<int> adj[n]
  - adj[u] stores a list of nodes which are adjacent to node u.
Matrix v.s. List

◇ Checking if two nodes are directly connected:
  ▪ Matrix: $O(1)$
  ▪ List: $O(n)$ worst, $O\left(\frac{m}{n}\right)$ average.

◇ Memory
  ▪ Matrix: $O(n^2)$
  ▪ List: $O(m + n)$
Outline

◊ Graphs
◊ Adjacency Matrix vs. Adjacency List
◊ **Special Graphs**
◊ Depth-first and Breadth-first Search
◊ Topological Sort
Special Graphs

◊ Tree
  ▪ A connected acyclic graph
  ▪ A connected graph with $n - 1$ edges
  ▪ An acyclic graph with $n - 1$ edges
  ▪ There is exactly one path between every pair of nodes
  ▪ An acyclic graph but adding any edge results in a cycle
  ▪ A connected graph but removing any edge disconnects it

◊ Bipartite Graph
  ▪ Separate into two groups of nodes such that the edges exist between these two groups only.

◊ Directed Acyclic Graph (DAG)
  ▪ Nodes have a partial ordering.
Outline

◊ Graphs
◊ Adjacency Matrix vs. Adjacency List
◊ Special Graphs
◊ Depth-first and Breadth-first Search
◊ Topological Sort
Depth-First Search

- **DFS(v):** visits all the nodes reachable from v in depth-first order
  - Mark v as visited
  - For each edge v → u:
    - If u is not visited, call DFS(u)
Example
Property

- In undirected graph
  - No cross edges, only tree-edges and back edges

Figure from Wikipedia Depth-first-search
Example Problems

◊ What is the minimum number of edges should be added to make an undirected graph connected?

◊ E.g.
◊ 5 nodes, 3 edges
◊ 1-2
◊ 1-3
◊ 2-3
◊ Then, the answer is 2.
Example Problems: Solution

- Figure out the number of components $N$
- The answer is $N - 1$
Breadth-First Search

◊ BFS(v): visits all the nodes reachable from v in breadth-first order
  ▪ Initialize a queue Q
  ▪ Mark v as visited and push it to Q
  ▪ While Q is not empty:
    • Take the front element of Q and call it w
    • For each edge w → u:
      – If u is not visited, mark it as visited and push it to Q
Example
Example Problems

◊ The distance from node $u$ to node $v$ are defined by the minimum number of edges should be traversed from $u$ to $v$.

◊ Find the furthest node from node 1.

◊ E.g. Answer is 3
Example Problems: Solution

- Depth of the BFS tree
Outline

◊ Graphs
◊ Adjacency Matrix vs. Adjacency List
◊ Special Graphs
◊ Depth-first and Breadth-first Search
◊ Topological Sort
Topological Sort Problem

◊ Input: a DAG $G = (V, E)$
◊ Output: an ordering of nodes such that for each edge $u \rightarrow v$, $u$ comes before $v$
Example

◊ Possible Orders:
   - 4,1,6,2,3,7,5
   - 4,6,3,1,2,5,7
   - ...

[Diagram of a network with nodes 1 to 7 and arrows indicating connections]
Topological Sort

◊ Key Idea
  • Any node without an incoming edge can be the “first element”
Topological Sort

- Precompute the number of incoming edges $\text{deg}(v)$ for each node $v$
- Put all nodes $v$ with $\text{deg}(v) = 0$ into a queue $Q$
- Repeat until $Q$ becomes empty:
  - Take $v$ from $Q$
  - For each edge $v \rightarrow u$:
    - Decrement $\text{deg}(u)$ (essentially removing the edge $v \rightarrow u$)
    - If $\text{deg}(u) = 0$, push $u$ to $Q$
- Time complexity: $\Theta(n + m)$
Example Problems

◊ $N$ people
◊ Have known $M$ relations that
  ▪ $u$ is strictly higher than $v$
◊ Check whether someone are lying.
Questions?