Graphs and basic traversals

Lecture 11

Akshar Varma

24–26 July, 2023

CS3000 Algorithms and Data

Introduction to Graphs

Problems on Graphs

Traversals

Bipartite Graphs and Matchings

Directed Graphs

Shortest Paths

Minimum Spanning Trees (MSTs)

Hard Graph Problems

1. Introduction to Graphs

Graphs

- A graph G models relationships between objects.
- The set of vertices G represents these objects.
- Edges E between vertices represent relationships.
- Graph is defined by $G = (V, E)$
- Examples, try to determine what are vertices and what are edges:
  Family trees, road networks, subway maps, internet routing, recommendation engines, disease spread, compiling code, folder structure, code dependencies, etc.

Types of Graphs

- **Directed vs. Undirected:** Edges have a direction or not. (u, v) vs. $\{u, v\}$
- **Cyclic vs. Acyclic:** Are there cycles in the graph?
- **Bipartite:** Split vertices into two such that no edges within parts.
- **Trees:** Undirected and has no cycles.
- **Rooted vs. Unrooted Trees:** A special vertex "root"; edges oriented towards/away
- **DAGs:** Directed Acyclic Graphs
- **Weights vs. unweighted:** Edges (sometimes but rarely vertices) have weights.

More Terminology

- **Degree of vertex v:** Number of edges “incident” on it. In-degree/Out-degree.
- **Neighbors of vertex v:** Set of vertices which have edges to/from v.
- **Paths:** Sequence of vertices with edges between them.
- **Cycles:** Path that visits a vertex twice.
- **Leaves in trees:** Orient edges away from root. Leaf has no edge going out.
- **Ancestors and Descendants:** Orient edges away from root.
- **Connectedness:** Path from every vertex to every other vertex.
- **Subgraphs:** $G = (V, E) \longrightarrow G' = (V', E')$ such that $V' \subseteq V$, $E' = \{\{u, v\} | u, v \in V'\}$
- **Complete graphs:** All possible edges exist.

- The sum of degrees in an undirected graph must be even
- Coloring; show a planar graph which needs 4 colors
- **Eulerian Path/Cycle:** Visit every edge exactly once.
- Trees:
    1. Connected, acyclic, undirected
    2. Unique path between any two vertices.
    3. +1 edge adds cycle
    4. -1 edge disconnects
- $\Big(\text{Min degree 2} \implies \text{cycles}\Big) \iff \Big(\text{Trees} \implies \exists v \in V, deg(v) = 1\Big)$

2. Problems on Graphs

Problems on Graphs

- Ordering vertices
  1. For traversal: Depth first search and Breadth first search
  2. For need of an ordering: Topological sorting, Hamiltonian paths
- Selecting shortest paths: Dijkstra and Bellman-Ford
- Maxim(um|al) Matchings
- Finding subgraphs with certain properties. Or determining if any exist:
  1. Minimum Spanning Trees
  2. Cycles (also bipartitieness)
  3. Connected Components
  4. Independent Set and Cliques
  5. Coloring vertices.

3. Traversals

Traversals

- We'll look at two separate ways of traversing through every node in the graph.
- Depth First Search (DFS):
  Descendants before Siblings
- Breadth First Search (BFS):
  Siblings before Descendants
- We'll need to know stacks and queues to properly implement these.
- Stacks and Queues will track things we've skipped and we'll come back to those later.

Data Structures Recap

- Binary Search Trees (BSTs)
- Priority Queues/Binary Heaps (visualized as trees)
- Stacks
- Queues

Depth First Search (DFS)

- Explain idea of continuing down descendants before going through siblings.
- Show on board. First with a tree/DAG, then a general graph.
- Why Stacks?
- Show demo.
- Application: Connectedness, Topological sorting. Many more.

Breadth First Search (BFS)

- Explain idea of visiting all siblings before going to descendants.
- Show on board. First with a tree/DAG, then a general graph.
- Why Queues?
- Show demo.
- Application: Connectedness, Bipartiteness/2-Colorability. Many more.

4. Bipartite Graphs and Matchings

Bipartite Graphs and Matchings

- Bipartiteness
- 2-Colorability $\iff$ Bipartite
- No odd cycles $\iff$ 2-colorable
- Matchings and maximum matchings
- Maximum matching algorithm deferred until Network Flows are covered.

5. Directed Graphs

Directed Graphs

- All algorithms until now will continue to work even for directed graphs (Digraphs).
- No algorithm assumed undirected edges hence it should work for digraphs.
- However, digraphs are more general than undirected graphs.
  Every undirected graph is a digraph such that $(u, v) \in E \iff (v, u) \in E$.
- So there are some problems that work differently for digraphs.

Topological Sorting of DAGs

- **Kahn's algorithm:**
  1. Pick every vertex without incoming edges, they'll be first.
  2. For every such vertex u, remove outgoing edges (u, v).
  3. If the endpoint v has no incoming edges, it can now be added to the list.
  4. Keep going until every vertex included.
  5. If at the end edges remained and something wasn't added, then there was a cycle.
  6. Otherwise return list.
- **DFS based:** Perform DFS, if leaf node or if all descendants visited then prepend the node to list.

6. Shortest Paths

Shortest Paths

We'll look at finding shortest paths in graphs under a few different circumstances:

- (Un)directed graph with no weights on edges.

  BFS!
- (Un)directed graph with positive weights on edges.

  Dijkstra's Algorithm: Our first greedy algorithm
- (Un)directed graph with any weights on edges but no negative cycles.

  Bellman-Ford: DP once again (yes, same Bellman).
- All of the above are from a fixed single source s to all other nodes.
- There's also Floyd-Warshall for all-pairs shortest paths.

  Not covered, but closely related to:

  1. Finding transitive closures of relations (representable as directed graphs)
  2. Converting deterministic finite automata to regular expressions (you'll see in ToC).

- Initialize distances: source is zero, $d[s] = 0$; and infinity for rest, $d[v] = \infty$.
- Source is current node c, every node is "unvisited" at start.
- Compare $d[c] + w(c, u)$ with $d[u]$. If shorter, update $d[u]$.
- After all edges $\{c, u\}$ considered, mark c as visited.
- Stop if *done*: "destination" marked "visited".
- Else: new current node u is an unvisited u with smallest $d[u]$; loop.
- If smallest $d[u]$ for unvisited is ∞ then disconnected. No path. Stop.
- Note: "smallest" $d[u]$ can be found quickly with priority queues/heaps.

Bellman-Ford Algorithm

- Similar initialization: source zero, $d[s,i]=0$; and infinity for rest, $d[v,i]=\infty$.
- i will represent how many edges are used in the path from s to node v.
- $d[v,i]$ will store shortest path from s to v using at most i edges.
- Recurrence:

$$d[v,i]=\begin{cases}0 & i=0, v=s\\ \infty & i=0, v\neq s\\ \min_{(u,v)\in E}\left(d[v,i-1], d[u]+w(u,v)\right) & \text{else}\end{cases}$$

- Keep doing this until $i=|V|-1$.
- An update when $i=|V|$ will mean cycles; if it happens, report negative cycle.
- Space improvement: Only track $d[v]$ since we only need $i-1$ to fill i.

7. Minimum Spanning Trees (MSTs)

Minimum Spanning Trees (MSTs)

- **Spanning:** Every vertex is included.
- **Tree:** No cycles, connected, undirected.
- **MST:** A spanning tree with minimum possible sum of edge weights.
- Many meta uses in graph algorithms. Invoked as subroutines for many problems.
- Other applications:
    1. Heavily used in computer networks (broadcasting for example).
    2. Taxonomy and classification (segments in images, clustering hierarchically, gene expressions)
    3. (Telecommunication | transport | water supply | electrical) networks.

Prim's MST Algorithm

Prim's Algorithm

- Start from some vertex.
- Build up a tree by repeatedly adding the lowest weight edge from tree to non-tree.
- Repeat until all vertex picked up.

- Priority Queues/Heaps to keep picking up lowest weight edge.

Kruskal's Algorithm

- Pick smallest weight edge.
- Need to check if an edge connects within the same tree or separate components.
- Repeat until all vertex in the tree.

- Union Find data structure to determine if same tree or not.

Reverse-Delete Algorithm (anti-Kruskal)

- Repeatedly remove largest weight edge as long as graph stays connected.

Cuts, Cutsets, and Proving MST Algorithm Correctness

- A **cut** in a graph: Split vertices into S and $V \setminus S$.
- **Cutset**: Subset of E such that one endpoint is on S and one isn't.
- In other words, a "crossing" edge.
- Algorithm proofs of correctness all use the above, specifically:
  Consider the intersection of a cycle with a cutset.
- Min weight edge must be in MST; Max weight edge must not be in MST.

8. Hard Graph Problems

Hard Problems

- Independent Set, Clique, Vertex Cover.

  Independent Set can be solved using DP on trees.
- Hamiltonian Path/Cycle, Traveling Salesman Problem (TSP).

  TSP has exponential time DP algorithm.
- 3-Coloring

  Greedy algorithm for $\Delta + 1$ coloring. Δ is max degree of graph.
- Graph Isomorphism: Are these two graphs the same if we just relabel vertices?