SoC 2023: Competitive Programming

Week-3: Graphs

Mentor: Virendra Kabra

Summer 2023

Contents

This discussion roughly follows Chapter-7 from an updated version of the handbook.

1 Terminology

A graph is a structure consisting of *nodes* (*vertices*) and *edges*.

Fig. 7.1 A graph with 5 nodes and 7 edges

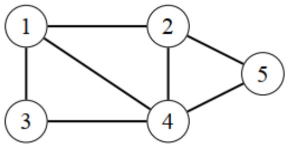


Fig. 7.2 A path from node 1 to node 5

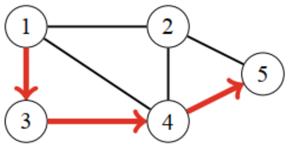


Fig. 7.3 A cycle of three nodes

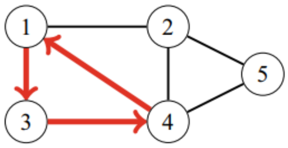


Fig. 7.4 The left graph is connected, the right graph is not

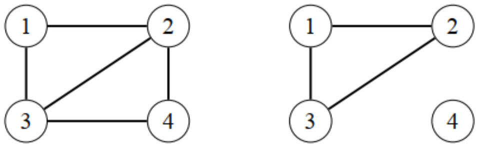


Fig. 7.5 Graph with three components

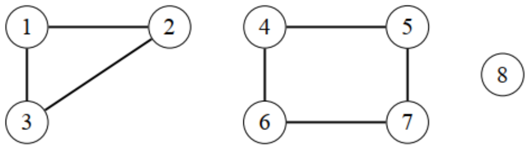


Fig. 7.6 A tree

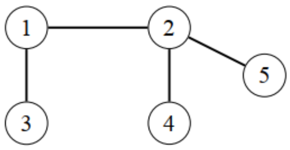


Fig. 7.7 Directed graph

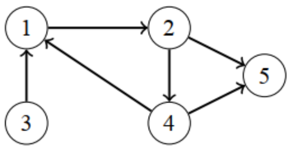
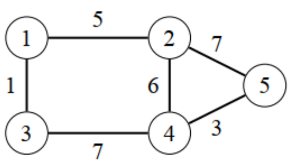
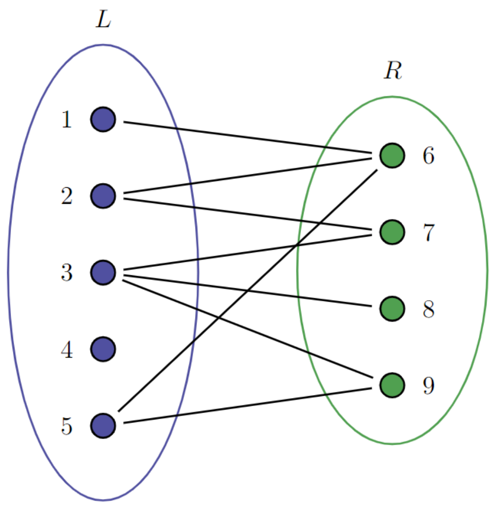


Fig. 7.8 Weighted graph



- Graph G is represented as (V, E). Set of vertices V, set of edges E. The first graph has $V = \{1, 2, 3, 4, 5\}$ and $E = \{(1, 2), (1, 3), (1, 4), (2, 4), (2, 5), (3, 4), (4, 5)\}$.
- Adjacent vertices (neighbors): Vertices connected with an edge
- Degree of vertex v: Number of neighbors of v. For directed graphs, *in-degree* and *out-degree* are defined.
- Path from a to b: Represented with a sequence of vertices $v_0 = a, v_1, \ldots, v_{k-1}, v_k = b$ such that v_i and v_{i+1} are neighbors. The length of a path is the number of edges in it. Path $u_0, \ldots, u_k$ is a cycle if $u_0 = u_k$.
- A graph is *connected* if there is a path between any two vertices. Connected parts of a graph are called *connected components*. For example, a connected graph has a single connected component.
- Tree: A **connected** graph with **no cycles**. Properties:
  - A tree on n nodes has exactly $n - 1$ edges.
  - Any connected graph on n nodes and $n - 1$ edges is a tree.
  - Any graph on n nodes with less than $n - 1$ edges is disconnected. Removing $k \leq n - 1$ edges from a tree gives $k + 1$ connected components.
  - There exists a *unique* path between any two vertices.
- Directed graphs have directed edges. For such graphs, edges (a, b) and (b, a) are not the same.
- Weighted graphs have weights associated with edges. For example, distance between two cities (nodes).
- Some special classes of graphs
  - Simple graphs: Do not have multiple edges between the same pair of nodes, and do not have self loops. For such graphs, $0 \leq |E| \leq \binom{|V|}{2}$. A *complete* graph has edges between all pairs of vertices. In later sections, we deal with simple graphs only.
  - Bipartite graphs:



2 Representation

Simple graph $G = (V, E)$ with $|V| = n$ and $|E| = m$.

- Adjacency list
- Adjacency matrix. Requires $O(n^2)$ space.
- Edge list

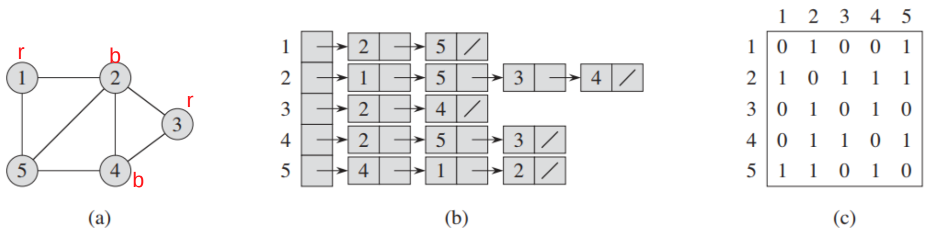


Figure 22.1 Two representations of an undirected graph. **(a)** An undirected graph G with 5 vertices and 7 edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

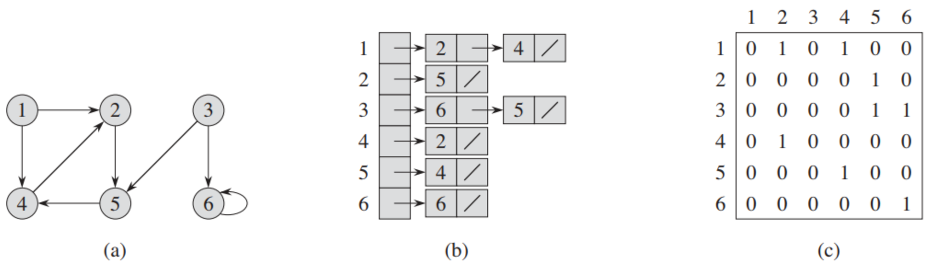


Figure 22.2 Two representations of a directed graph. **(a)** A directed graph G with 6 vertices and 8 edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

Check code files for an implementation.

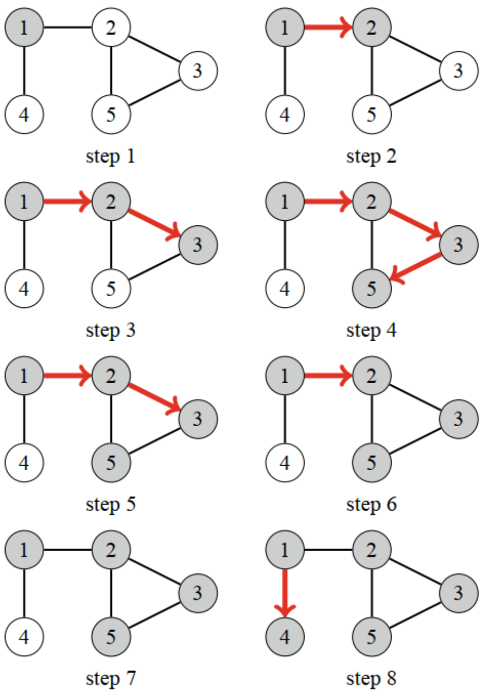
3 Traversal

We discuss DFS and BFS. Given a starting node, all nodes are *visited* in some order. Each node and edges is visited once, giving a complexity $O(|V| + |E|)$ for both traversals. Code files contain an implementation.

3.1 Depth-First Search

Follows a single path in the graph till new nodes are found. Then returns to previous nodes and begins exploration of other parts of the graph. Usually implemented with recursion.

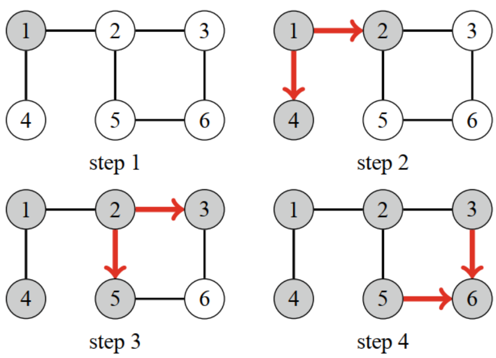
Fig. 7.13 Depth-first search



3.2 Breadth-First Search

Visits nodes in increasing order of their distance from the starting node. A queue is maintained.

Fig. 7.14 Breadth-first search



For disconnected graphs, these can be called on one vertex in each component to traverse the entire graph.

3.3 Applications

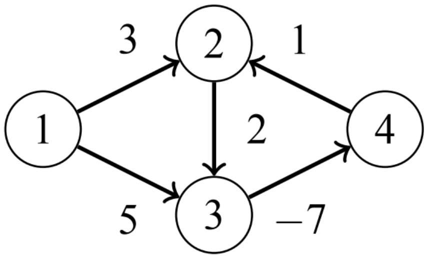
Consider undirected graphs.

- Check if graph is connected: Perform traversal from one node. If some node is not visited, the graph is disconnected.
- Detect cycle: A visited node (other than the current node's parent) is encountered in traversal.
- Check is graph is bipartite: Observe from the image that G can be 2-colored. Perform traversal from a node s; color it 0 (for example, `visited[s] = 0`). Alternately assign colors 1 and 0. If a neighbor is already colored, and with the same color as current, the graph is not bipartite.

Check code files later.

4 Shortest Paths

Given graph G and vertices a and b, find the shortest path between them. For unweighted graphs, BFS suffices.
Now we deal with weighted graphs. Note that for a graph with negative cycles, shortest paths may not be defined, as their lengths can be $-\infty$. For example, shortest path from 1 to 4 below.



4.1 Single Source Shortest Paths

Find shortest paths from a given *source* vertex to all other vertices.

4.1.1 Dijkstra's Algorithm

It is required that there be no negative weight edges in the graph. A priority queue of nodes is maintained, to get the unvisited node with smallest current distance. Complexity $O(m \log n)$.

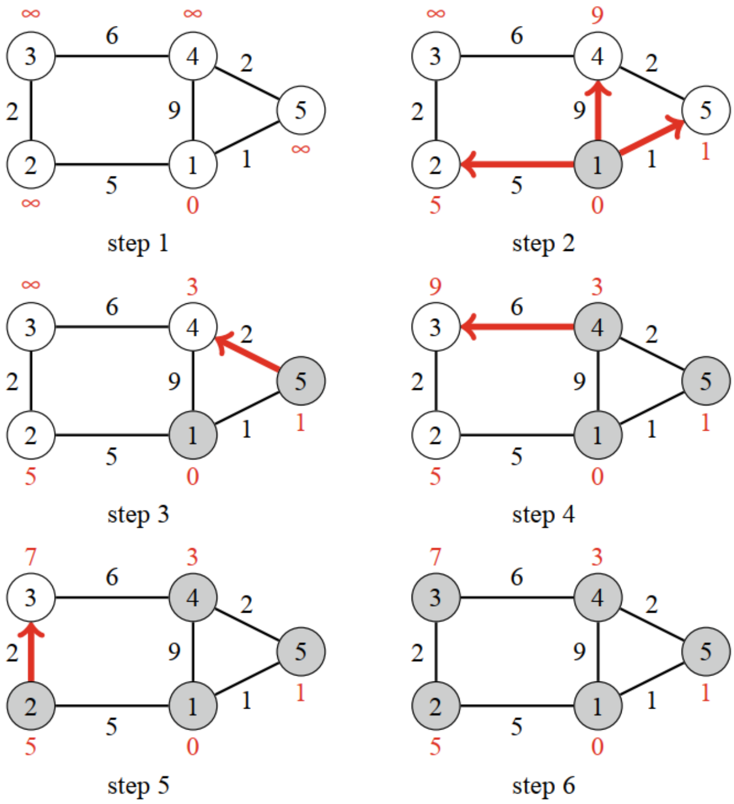


Fig. 7.20 Dijkstra's algorithm

4.1.2 Bellman Ford Algorithm

Allows negative weight edges, and detects negative cycles. It consists of n rounds, and in each round the algo goes through all edges, attempting to reduce current smallest distances. Implemented with edge-list representation. Complexity: $O(nm)$.

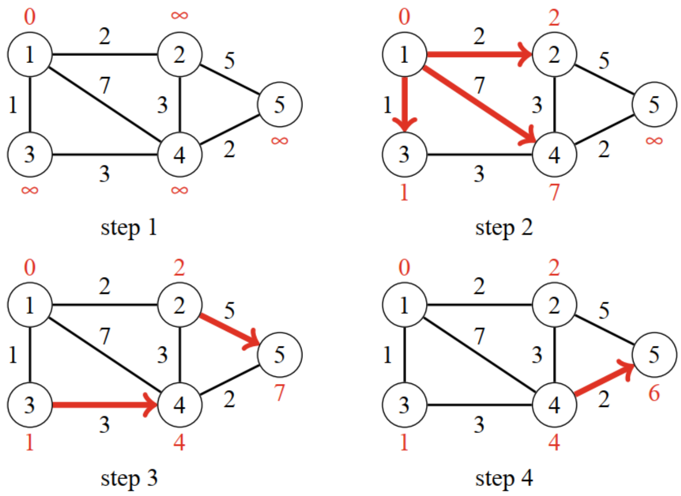


Fig. 7.18 The Bellman–Ford algorithm

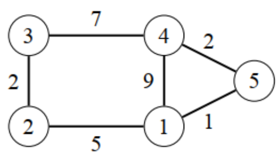
If the last round reduces any distances, there is a negative cycle.

4.2 All Pairs Shortest Paths

Find shortest paths between all pairs of vertices.

4.2.1 Floyd Warshall Algorithm

An adjacency matrix with (smallest) distance entries is maintained. The algorithm consists of consecutive rounds, and on each round, it selects a new node that can act as an intermediate node in paths from now on, and reduces distances using this node. Complexity: $O(n^3)$.

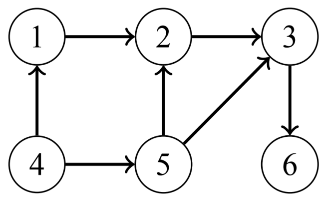


$$\begin{bmatrix} 0 & 5 & \infty & 9 & 1 \\ 5 & 0 & 2 & \infty & \infty \\ \infty & 2 & 0 & 7 & \infty \\ 9 & \infty & 7 & 0 & 2 \\ 1 & \infty & \infty & 2 & 0 \end{bmatrix} \begin{bmatrix} 0 & 5 & \infty & 9 & 1 \\ 5 & 0 & 2 & \mathbf{14} & \mathbf{6} \\ \infty & 2 & 0 & 7 & \infty \\ 9 & \mathbf{14} & 7 & 0 & 2 \\ 1 & \mathbf{6} & \infty & 2 & 0 \end{bmatrix} \begin{bmatrix} 0 & 5 & \mathbf{7} & 9 & 1 \\ 5 & 0 & 2 & 14 & 6 \\ \mathbf{7} & 2 & 0 & 7 & \mathbf{8} \\ 9 & 14 & 7 & 0 & 2 \\ 1 & 6 & \mathbf{8} & 2 & 0 \end{bmatrix}$$

Simulation starting with nodes 1 and 2.

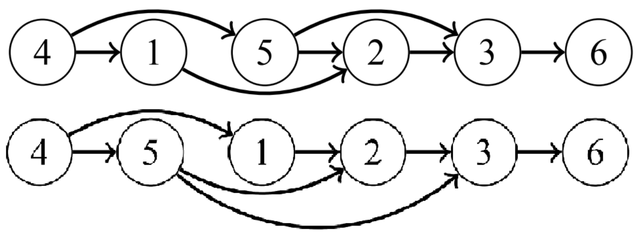
5 Directed Acyclic Graphs (DAGs)

Directed graph with no directed cycles.



5.1 Topological Sort

- An ordering of vertices such that if there is a directed edge (u, v), then u appears before v in this ordering. Some orderings for the above DAG:



Note that all edges go from left to right.